


Tracing “Toxicity” Through Code: Towards a Method of Explainability and Interpretability in Software

David M. Berry <d_dot_m_dot_berry_at_sussex_dot_ac_dot_uk>, University of Sussex  <https://orcid.org/0000-0002-7737-5586>

Abstract

The ubiquity of digital technologies in citizen’s lives marks a major qualitative shift where automated decisions taken by algorithms deeply affect the lived experience of ordinary people. But this is not just an action-oriented change as computational systems can also introduce epistemological transformations in the constitution of concepts and ideas. However, a lack of public understanding of how algorithms work also makes them a source of distrust, especially concerning the way in which they can be used to create frames or channels for social and individual behaviour. This public concern has been magnified by election hacking, social media disinformation, data extractivism, and a sense that Silicon Valley companies are out of control. The wide adoption of algorithms into so many aspects of peoples’ lives, often without public debate, has meant that increasingly algorithms are seen as mysterious and opaque, when they are not seen as inequitable or biased. Up until recently it has been difficult to challenge algorithms or to question their functioning, especially with wide acceptance that software’s inner workings were incomprehensible, proprietary or secret (cf. open source). Asking why an algorithm did what it did often was not thought particularly interesting outside of a strictly programming context. This meant that there has been a widening explanatory gap in relation to understanding algorithms and their effect on peoples’ lived experiences. This paper argues that Critical Code Studies offers a novel field for developing theoretical and code-epistemological practices to reflect on the explanatory deficit in modern societies from a reliance on information technologies. The challenge of new forms of social obscurity from the implementation of technical systems is heightened by the example of machine learning systems that have emerged in the past decade. A key methodological contribution of this paper is to show how concept formation, in this case of the notion of “toxicity,” can be traced through key categories and classifications deployed in code structures (e.g. modularity and layering software) but also how these classifications can appear more stable than they actually are by the tendency of software layers to obscure even as they reveal. How a concept such as “toxicity” can be constituted through code and discourse and then used unproblematically is revealing in relation to both its technical deployment but also for a possible computational sociology of knowledge. By developing a broadened notion of explainability, this paper argues that critical code studies can make important theoretical, code-epistemological and methodological contributions to digital humanities, computer science and related disciplines.

Introduction

In the past decade, due to the perceived lack of accountability of algorithmic systems, particularly automated decision systems, a new explanatory demand has crystallized in an important critique of computational opaqueness and new forms of technical transparency called “explainability.” We see this, for example, in challenges to facial recognition technologies, public unease with algorithmic judicial systems and other automated decision systems. There have been new regulatory attempts to capture some of the ideas that stem from explainability such as the *Algorithmic Accountability Act 2022* in the US, and the *General Data Protection Regulation 2016/679* (GDPR) in the European Union. These forms of regulation mandate a requirement for a user of algorithms to be able to seek a representation (or “explanation”) of an algorithm used in an automated decision by a computer system and that the developers should provide one (see [Goodman and Flaxman 2017], [Lum Chowdhury 2021], [Selbst Powles 2017], cf [Wachter, Mittelstadt,

Floridi 2017]). These are important issues and are changing the way in which algorithms are designed and implemented.^[1]

This public concern has been magnified by election hacking, social media disinformation, data extractivism, and a sense that Silicon Valley companies are out of control. The wide adoption of algorithms into so many aspects of peoples' lives, often without public debate, has meant that increasingly algorithms are seen as mysterious and opaque, when they are not seen as inequitable or biased. Up until recently it has been difficult to challenge algorithms or to question their functioning, especially with wide acceptance that software's inner workings were incomprehensible, proprietary or secret (cf. [Berry 2008] regarding open source). Asking why an algorithm did what it did was often not thought to be particularly interesting outside of a strictly programming context. This meant that there has been a widening explanatory gap in relation to understanding algorithms and their effect on peoples' lived experiences.^[2]

However, in this paper rather than focus on the regulative aspect, I want to use the insights of critical code studies to explore how the code itself might usefully be analysed in situ, rather than through a secondary representation created by an "explainable" black-box.^[3] Explainability is a key new area of research within the fields of artificial intelligence and machine-learning and argues that a computational system should be able to provide an explanation for an automated decision. This has become known as the problem of explainability for artificial intelligence research and has led to the emergence of the subfield of Explainable Artificial Intelligence (XAI) (see also [DARPA n.d.], [Sample 2017], [Kuang 2017]). There has been an important theoretical distinction within this field of explainability between the notions of *explanation* and *interpretation*. An interpretable system can be understood as designed as a human-readable system from the bottom up. This often uses "simple" models, straightforward calculations or filtering which can be communicated easily. In contrast, an explainable system is a system that incorporates a secondary automated system, usually using machine learning, to machine-read and model the primary system, and thereby to provide an explanation of how this system works. Explainability has therefore emerged as a term that usually seeks to close an explanatory gap by means of responses generated by technology itself. As Rudin argues, "the field of interpretability / explainability / comprehensibility / transparency in machine learning has strayed away from the needs of real problems.... Recent work on explainability of black boxes – rather than interpretability of models – contains and perpetuates critical misconceptions that have generally gone unnoticed, but that can have a lasting negative impact on the widespread use of machine learning models in society" [Rudin 2019, 2]. Indeed, I argue that the idea of a technological response to an interpretability problem is doomed to failure while explainability is understood through such narrow technical criteria and instead requires interdisciplinary approaches([Berry 2023]; see also [Dobson2021]).

The aim in this paper is to develop the insights developed through critical code studies which uses methods of interpretation and close reading of the code to contribute to re-thinking the notion of explainability by looking at a case study. In this paper I use *explainability* to gesture towards a method of tracing the underlying mechanisms and interactions within software towards an explanation of the functioning of the code. Within artificial intelligence, explainability is sometimes explicitly linked to automated processes of producing explanations, whereas interpretability is linked to human-readable code that is easily understood by a human programmer. In both cases an explanation is sought, so here I prefer to use the term *explainability* as a critical method for seeking an explanation, and highlight "automated explainability" where this is undertaken by software (Explainable AI). This can be usefully compared with *interpretability* as the use of a model for the code that is explicitly written to be easily read and understood by humans (Interpretable AI) (see [Rudin 2019]). Broadly speaking, explainability has tended to become an automated approach, and here I want to argue that if it were re-articulated through the concept of interpretability it could become a strong analytical method for reading computer code critically. I also hope to show that critical code studies and explainability are ways of thinking about the digital and computation that would be extremely useful for digital humanities more widely but also for STEM education and computer science more generally (see also [Marino 2020, 230]).^[4]

1. Theoretical Context

In this paper, therefore, I seek to widen explainability's applicability through Critical Code Studies and in doing so, to create connections to ideas of explanation, the paratextual^[5] of computer source code, and the value of close-reading

code as an explanatory approach. By the paratextual elements I am gesturing to the importance of a more capacious notion of source code that includes documentation, commentary, blog-posts and even social media related to the code. In this case I look at the “WIT Toxicity Text Model Comparison” (What-If-Tool) developed by Jigsaw and Google (both units within Alphabet) that aims to classify “toxic text” in a dataset and which contains a remarkable normative structure incorporated into its algorithms.^[6]

Within the code that I examine in this paper, the notion of “toxic” and “toxicity” and the relation of these words to their deployment in software are crucial for this analysis. By using approaches from critical code studies, the aim is to help explain how a machine learning model can classify “toxicity” by tracing the concept through layers of software, the so-called software stack, by close attention to the source code which serves as a document of its usage.^[7] That is that code can be made explainable by tracing concepts through a close reading of the source code.

Critical code studies has been developed and supported by a range of scholars working particularly at the level of code to attempt to apply humanistic interpretative approaches to the study of the source code in computer systems.^[8] One of the latest examples of this is Mark Marino’s *Critical Code Studies* which includes a number of examples of reading source code in practice [Marino 2020]. He writes,

in our digital moment, there is a growing sense of the significance of computer source code. It has moved beyond the realm of programmers and entered the realm of mainstream media and partisan political blogosphere. Those discussing code may not be programmers or have much fluency in the languages of the code they are quoting, but they are using it to make, refute, and start arguments. There is also a growing sense that the code we are not reading is working against our interests. [Marino 2020, 3]

Marino’s approach foregrounds the use of case studies and close reading of the code to develop a number of strategies drawn from the humanities for understanding code. Indeed, he argues critical code studies “was born of the humanities” [Marino 2020, 228]. Essentially Marino foregrounds the importance of “understanding” and exegesis for this approach [Marino 2020, 20, 237].

I would like to start with a discussion of this theoretical approach inasmuch as when we come to understand computer source code we are immediately faced with a dilemma in terms of a reading which is attentive to this interpretative dimension together with the explanatory aspect. In a sense I want to present a double aspect theory in relation to computer source code, in particular the way in which it is both a casual-deterministic mode of what we might call mechanical agency, and its interpretative mode of textuality. Marino highlights the understanding applied through approaches which privilege the interpretative dimension, but I argue it is crucial that we are able to move between these two aspects (explanatory/ interpretative) if we are to provide a full account of computer code. This is also a useful distinction to make if we are to make sense of the current debates over explainability. We might note that a broadly interpretative approach which focuses on case studies may lose the explanatory aspect of the mechanical operation of computer code. In other words, it might lose the value of explaining why code has the structure it does and why a particular computer language has the value it does for its programmers, which could be its effective or expressive potentialities, for example.

In contrast I would like to use a *realist* notion of computer code, which involves an attempt to describe the real structures, entities, and processes that make up computer code and which exist independently of our descriptions of them. So, we should be attentive to the tendencies that emerge from the causal power of entities, structures, and mechanisms. That is, to focus on the real, operative mechanisms which may or may not produce observable results. This is because computer source code consists of a number of sub-systems which draw on a “multiplicity of casual mechanisms at work at any time” and can be explained through a broadly realist explanatory framework [Outhwaite 1987, 175]. This opens up a number of useful complements to interpretative approaches to understanding code because it allows reference to unobservable entities which go beyond our direct experience, and which therefore help to demonstrate the existence of non-obvious determinations of observable events within code in operation, or which can be inferred from the inclusions or references of programmers to paratexts in the code. This is not to descend into metaphysics, indeed one should follow the advice of Whitehead and avoid “misplaced concreteness,” that is, to too

easily assume that concepts are entities that are unproblematically in existence. Here we might follow [Hacking 1983, 27] and distinguish between “realism about theories from realism about entities” [Outhwaite 1987, 178]. As Outhwaite explains, “to put it a bit crudely, true theories state what is the case. A realism about entities simply asserts that some thing or things exist” [Outhwaite 1987, 178]. In avoiding the notion of determinate entities we instead focus on the provisional understanding of computational structures and mechanisms. So that the practices of computer programmers and users cannot be understood except in terms of the computational structures within which they participate. This is to highlight how the explanatory dimension of computer code can usefully contribute to “filling out” interpretative textual approaches to understanding code.

In terms of understanding a particular case study of computer code, I therefore argue that we would need to include in any account of source code the explanatory layers that might be found upstream from the code under investigation. Not so much black-boxes as obscure networks of code which are formed from the social and economic factors (or paratextual elements) which surround code. This includes both networks in a technical sense, as the code is structured into self-standing objects, functions and applications, but also in a social sense, especially in terms of the social networks that are important in creating and maintaining the code in operation. These can be thought of as the infrastructures of code. These are instantiated within computational frameworks, operating system structures, data structures, language constraints, hardware specificities, or any number of wider sociological structures that might be not observable directly but which are sedimented within code downstream from these larger structural frameworks or the code situation with which we are working. Clearly, in this case “the nature of the object should determine the most appropriate methods of investigation” in terms of the whether one attempts to determine micro or macro studies in relation to it [Outhwaite 1987, 182]. This brings forward the importance of the selection made as a researcher and the reflexivity that should be foregrounded in any analysis of critical code study.

Additionally, it is crucial to understand the conditions of action and the goals towards which a particular piece of code is aimed. The “conditions” or “goals” may be defined as givens. If directed by a project or work situation these may often be fixed, unexaminable or relatively unchangeable. These conditions or goals defined as givens will therefore variously limit and constrain the resultant focus of the code and may impact directly the language choice, framework, approach and dependencies used. They can usually be discerned in the surrounding documentation of code or in the comments where programmers often highlight the direction of processing and its intended function. Indeed, sometimes programmers may express their feeling or complain about the code quite explicitly if they feel constrained by the framework or language choice foisted upon them (see [Berry 2011, 69]). Some organisations also have a “house style” of code, which guides the choice of words representing variables, capitalisation, functional relations (e.g., internal libraries and dependencies), commenting practices and even the formatting of the code. This may be an established or published naming and coding standard within an organisation. But any bureaucratic discipline laid on the coding practices and source-code format is grounded in the expectation of resistance. Indeed, source code can sometimes reveal the pressures between technical practice and bureaucratic limitation, most notably expressed in the comments sections, or sometimes in variable naming. However, if the code produced is too much at variance with the style or practices of an organisation then there can be a “return of the repressed” in which the repressed element of the freedom of the programmer in a “house style” may manifest itself in idiosyncratic code or resistance expressed in the coding (see [Berry 2011, 68–70]). Indeed, the fundamental repression on which bureaucratic organisations are based on, in terms of standardisation and process management, may surface when programmers stray too far from the limits set by organisations and result in heavy-handed responses including possible expulsion of the programmer from a project, or even their termination of employment. Hence, to avoid conflict in technical work such as programming a certain degree of autonomy tends to be present and the bureaucratic concern with the software may leave the technical process to the programmer and confine judgement to the appraisal of the final code product. By the means of assessment by result, normal bureaucratic management can be maintained over software development. This is helpful for critical code studies as within the source code there can often be traces left behind which give clues as to the way in which a particular algorithm was formulated, the assumptions or limitations of the code, and occasionally the conflictual aspects of code-writing which are documented in comments.

Within the source code although we can sometimes observe this conflict play out, but more often code is presented in

the form of a prosaic matter-of-factness stripped of its individuality. It has, so to speak, been painted grey in grey, as Hegel observed about uncreative philosophy. This common style in programming, which we might call “coding from nowhere” is seen where code is presented as apolitical, disinterested and placed “above struggle.” Many examples of this can be seen in open source and free software projects where the values of the project may be more easily observed in the code, documentation and debates that are often freely available to examine on mailing lists and social media (see [Berry 2008]). As we will see in the case of the code later examined in this paper, although a technical project might aim to align with the interests, values and goals of society in terms of notions of communicative discussion free from domination, which in this case they define as “toxicity,” close reading the code demonstrates the devotion of the programmers to the development of means and instruments. By focussing on technical means they thereby repress the ends and morality they purport to support. Indeed, in this case the code acts to automate the processes of judgement over the “toxicity” of online discussions creating abstractions and a misplaced concreteness which results in a simplistic binary characterisation of discourse. That is, the models either find a comment toxic or not toxic. This process of technical hollowing out of the ends is also apparent in the processes of debiasing one of the text models, by application of a set of weightings on crowd-sourced judgements of self-declared identity characteristics, in an attempt to normalise model judgements. The technical regress this opens up is clear when one asks “who debiases the debiasers?”^[9] A question, incidentally that the system builders do not tend to ask.^[10] To paraphrase György Lukács, who sarcastically lampooned the debates of the Frankfurt School philosophers in the 1930s who would build critical theories whilst enjoying a privileged economic condition, we might say that machine learning is similarly a beautiful hotel, equipped with every comfort, on the edge of an abyss, of nothingness, of absurdity.

2. Reading Code

I now want to turn from this theoretical and methodological discussion to look at how these insights might inform a realist critical code studies reading of code. The aim is to think about the way in which a notion of explainability can be incorporated into a reading of a source code corpus. Due to limitations of space, I narrow the focus to look at an explanation of how “toxicity” itself is deployed and understood in the code under review. To do this I look at the “What-If Tool toxicity text model comparison” text-processing project.^[11] This uses a tool developed by Google, called the “What-if Tool”, that allows a user to,

14

test performance in hypothetical situations, analyze the importance of different data features, and visualize model behavior across multiple models and subsets of input data, and for different ML fairness metrics. [WIT]

The “What-if Tool” offers a “range of interactive visualizations and guided explorations of a TensorFlow model, allowing developers to explore how their model interpreted its training data and how subtle changes to a given input would change its classification” [Leetaru 2019]. The WIT Toxicity Text Model Comparison (TTMC) uses this “What-if Tool” to “compare two text models from ConversationAI that determine sentence toxicity, one of which has had some debiasing performed during training” [TTMC 2021]. This is quite a complex piece of software, drawing on a number of libraries and data sources, and applied to the wikipedia comments dataset.^[12] This dataset is a tsv file which includes over “100k labeled discussion comments from English Wikipedia. Each comment was labeled by multiple annotators via Crowdfunder on whether it is a toxic or healthy contribution” [Wikipedia Talk Labels 2021].^[13] Essentially this is a dataset with labelling using a machine learning model trained by crowd-sourced using a simple machine learning classifier based on a set of 4053 people being asked whether a comment on Wikipedia contained a personal attack or harassment [Wulczyn, Thain, Dixon 2017, 2].^[14] This was enumerated in one of five sub questions for classification by the programmers, see figure 1.

15

Does the comment contain a personal attack or harassment?

- Targeted at the recipient of the message (i.e. you suck).
- Targeted at a third party (i.e. Bob sucks).
- Being reported or quoted (i.e. Bob said Henri sucks).
- Another kind of attack or harassment.
- This is not an attack or harassment.

Figure 1. The question posed to Crowdfunder annotators ([ToxicityQuestion 2017], [Wulczyn, Thain, Dixon 2017])

It is important to note that the machine learning model applied to the wikipedia comments dataset by Crowdfunder is different from the ones tested in the TTMC tool. Crucially it is the result of a different crowd-sourcing process and therefore the concept of “toxic” in the dataset is different to the concept of “toxic” used in the TTMC. Nonetheless the Crowdfunder toxicity classification is left as a trace, or data field, in the dataset tsv file. This Crowdfunder toxicity classification is then utilised in the TTMC tool as a static value so that it can be compared to the other models it compares. Although complex, understanding the different ways in which “toxicity” is being algorithmically produced as a value is an important part of making an explanatory account of this code.

16

The WIT Toxicity Text Model Comparison is hosted on the Google Colab or Colaboratory which is a free Jupyter notebook environment that runs in the cloud on a browser and stores its notebooks on Google Drive. Jupyter Notebook is a web-based interactive computational environment for creating and running notebook documents, which are browser-based “REPL” (Read–eval–print loops) containing “an ordered list of input/output cells and which can contain code, text, mathematics, plots and rich media” [Wikipedia 2021]. This environment allows the development of code in an interactive fashion and has become very popular as a way of doing data science and other programming activities which are exploratory. The ease with which a Jupyter notebook can work with data and code allows a program to be viewed, edited and run in real-time to test ideas. This means that it is an ideal environment for facilitating critical code studies projects.

17

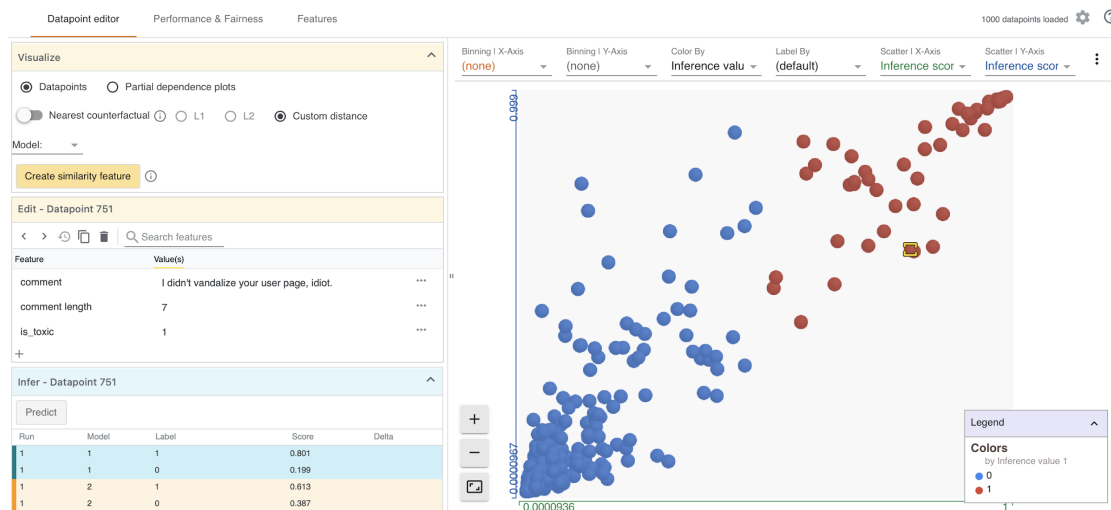


Figure 2. The What-If Tool Data Visualiser

The main part of the analysis of the WIT Toxicity Text Model Comparison compares two models drawn from ConversationAI, an “initiative to protect voices in conversation,” to apply to the Wikipedia comments dataset. ConversationAI claims to develop tools so that “globally, fewer people are silenced and more people are able to safely engage in good faith discussion online.”^[15] They further argue that their team “leads as an example of ethical practices

18

in building technology” [ConversationAI 2021a].^[16] ConversationAI is developed by Jigsaw and the Google Counter-Abuse Technology Team and hosted on GitHub.

The two main pre-trained models used in the analysis are `cnn_wiki_tox_v3_model.h5` and `cnn_debias_tox_v3_model.h5`.^[17] The latter has had debiasing performed during its training. Both models are convolutional neural network models, accessed using the Keras open-source software library (see [Liu Anci 2019] for a comprehensive explanation of generating models from sentence templates).^[18] CNNs work by modelling animal visual perception, and can therefore be applied to automatic recognition of patterns. CNNs are made up of multiple layers of individual software sensory neurons (so-called receptive fields, which are made up of clusters of these neurons). The word “convolution” comes from its use to describe a mathematical operation on two functions which produces a third function. These models were created for the Computefest 2019 conference using a set of Sentence Templates datasets developed by Jigsaw which have been “generated by plugging identity terms, occupations, and modifiers into a set of templates, e.g., ‘I am a <modifier> <identity>,’ to form test sentences.” They claim that “As only the identity term varies, examples using the same template — e.g., ‘I am a kind American’ and ‘I am a kind Muslim’ — should return similar toxicity scores. Scores that vary significantly may indicate identity term bias within the model” [Vasserman et al. 2021]. As can already be seen from this initial simple mapping of the software, there are a number of complex dependencies between the software being looked at, namely the WIT Toxicity Text Model Comparison, and the way in which it draws on a software toolkit, (the What-If Tool), other models (ConversationAI), Sentence Templates (Jigsaw), software libraries (Keras), and datasets shared on figshare (Wikipedia Talk Labels).

19

In developing a notion of explainability that considers the specificity of software system design and operation we can draw from Manovich’s principles of new media which he outlines as (1) numerical representation, (2) modularity, (3) automation, (4) variability and (5) transcoding [Manovich 2001, 27–48]. In reading the TTMC code I want to particularly focus on the principle of modularity to help trace concepts. Complex software such as the TTMC is often built on a number of layers which can be understood as akin to a building, with higher levels of the code resting on lower levels until eventually a key foundational level might be discerned. This forms part of the practice of well-structured modularity in code that enables software abstraction to be used whilst also simplifying complexity and creating more maintainable code bases. A key methodological practice I want to use here is the tracing of a fundamental organising concept used, often unproblematically in higher levels of the code, back to its original specification or construction. This is made easier in well-structured code as the different layers of the software can be understood as distinct and performing low-, mid-, or high-level processing functions. Using this modular principle in the interpretative practice can help explain the code functioning and provide a method for tracing conceptual usage within and across software.

20

In the case of TTMC, the code level at which the comparison tool is accessed directly makes use of a key framing concept and deploys it in its code operations which are then spread horizontally and vertically through the source code. Due to limitations of space I can only give limited textual examples from the source code, but my aim is to trace how toxicity becomes instantiated within the code, losing its nuance and context. By allowing the constellation of code, libraries and data formed around it to reify it into a thing that can be calculated and used to test other textual discourse we are able to see how a social concept becomes concretised into a technical concept of “toxicity.” The first place we see the introduction of this concept is within the form “Read the dataset from CSV and process it for model” in the code cells of the TTMC,

21

```
# Read the dataset from the provided CSV and print out information about it.
df = pd.read_csv(csv_path, names=csv_columns, skipinitialspace=True)
df = df[['is_toxic', 'comment']]
```

This code simply reads in the data from the wiki dataset and adds it to a dataframe(df).

22

```
label_column = 'is_toxic'
make_label_column_numeric(df, label_column, lambda val: val)
```

This is then labelled and turned into a numeric field as “is_toxic” is a value either 0 or 1. This is drawn from the

23

Wikipedia Comments Corpus and available on Figshare [Wikipedia Talk Labels 2021].^[19] It is important to note that in this corpus the problem of identifying personal attacks is treated as a binary text classification problem,

toxicity: Indicator variable for whether the worker thought the comment is toxic. The annotation takes on the value 1 if the worker considered the comment toxic (i.e worker gave a toxicity_score less than 0) and value 0 if the worker considered the comment neutral or healthy (i.e worker gave a toxicity_score greater or equal to 0). Takes on values in {0, 1}. ([Detox 2021], [ToxicityQuestion 2017])

We should note that the toxicity defined from the wiki dataset is already normalised to a binary true or false value and that the majority of the toxicity calculation in this dataset was created using the machine learning model described above. However, nowhere in this code level is toxicity defined or explained, to understand this key concept for the code we will need to widen our investigation to one of the key dependencies within the ConversationAI models and look to another project [ConversationAI 2021b]. By accessing the click through link to the source code in Github we find the underlying ConversationAI code [ConversationAI 2021a]. Within the file `unintended_ml_bias/new_madlibber/madlibber.py` we find,

24

```
self.__template_word_categories = set([])
self.__toxicity = set(["toxic," "nontoxic"])
```

As explained in a comment in the files `fat-star-bias-measurement-tutorial.ipynb` and `FAT_Star_Tutorial_Measuring_Unintended_Bias_in_Text_Classification_Models_with_Real_Data.ipynb`,

25

```
"Let's examine some rows in these datasets. Note that columns like toxicity and male are percent scores.\n"
```

```
** toxicity: this is the percentage of raters who labeled this comment as being toxic.\n"
```

However, the definition of toxicity, beyond a percentage value, is under specified and again we are required to dig deeper into the code dependencies where in the ConversationAI Perspective API we find a definition. However we might note the beginning of a slippage between the definition of toxicity as a thing and toxicity as a probability value. In the ConversationAI system "toxicity" is defined as "as a rude, disrespectful, or unreasonable comment that is likely to make someone leave a discussion" [Perspective 2021]. Toxicity's effects are understood as,

26

```
stop[ing] people from engaging in conversation and, in extreme cases, [it] forces people offline. We're finding new ways to reduce toxicity, and ensure everyone can safely participate in online conversations. [Jigsaw 2021]
```

Perspective API also helpfully documents how its model is able to classify text into a number of different outputs, all of which are interesting in and of themselves, but here it is the Toxicity value that we are investigating (see figure 2) (see also [Developers 2021]).^[20]

27

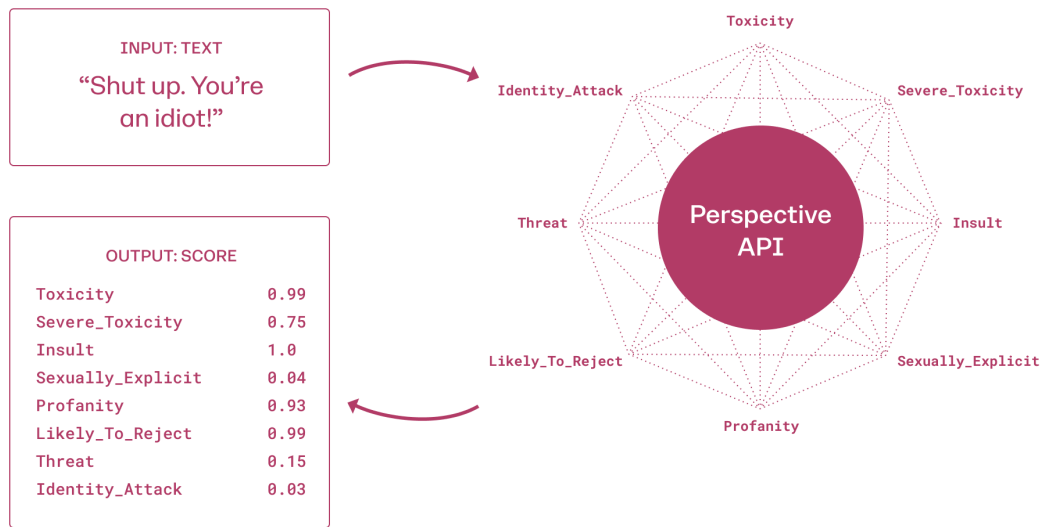


Figure 3. Overview of Perspective API [Perspective 2021]

Using the machine learning model the Perspective API returns what they call the “flagship Toxicity attribute” [Perspective 2021]. The further one digs into the ConversationAI documentation the more slippery the notion of toxicity becomes, from either 0 or 1 in the TTMC tool (and drawn from the Wikimedia dataset) to toxicity as a perception probability between 0 and 1. The documentation explains,

the model output is a probability. As such, a comment with a TOXICITY score of 0.9 is not necessarily more toxic than a comment with a TOXICITY score of 0.7. Rather, it’s more likely to be perceived as toxic by readers. [Perspective Model Cards 2021]

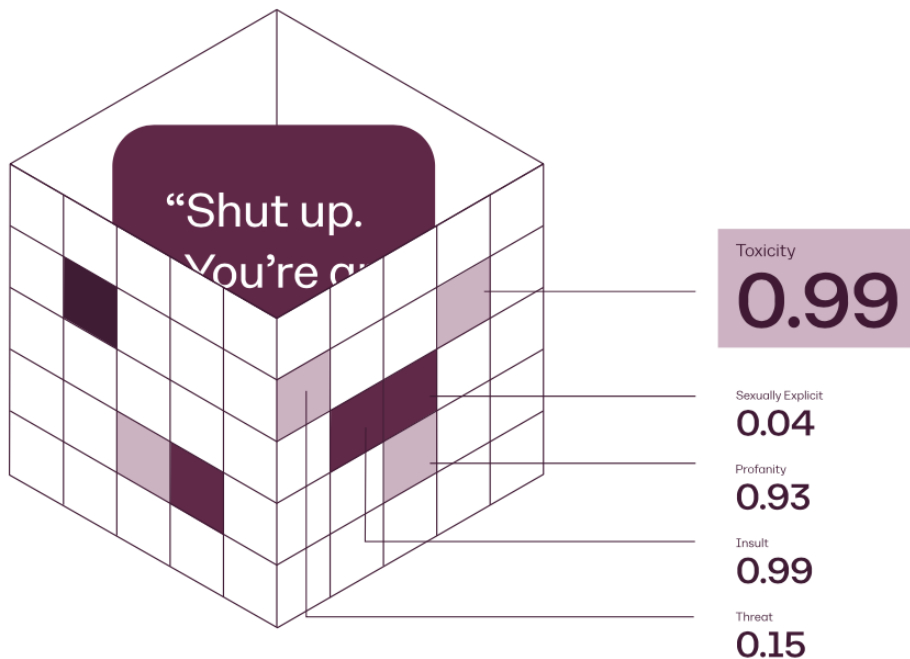


Figure 4. Toxicity Calculation [Perspective 2021]

As we move through the layers of the code, and indeed through the layers of meaning attached to the notion of “toxicity” we find that the inputs to the toxicity value eventually start to hit what we might call a humanistic level. That is, the actual source of the value is revealed as a calculation based on a set of human users asked to review and annotate a comment dataset.

29

Each comment is tagged by 3-10 crowdsourced raters from Figure Eight, Appen and internal platforms. The raters tag whether or not a comment contains an attribute (e.g., TOXICITY). We then post-process the tags to obtain labels corresponding to the ratio of raters who tagged a comment as toxic. For example, we label a comment as 0.6 for TOXICITY if 6 out of 10 raters tagged a comment as toxic. [Perspective Data 2021]

Here we are not given the actual datasets which underlie the model, and we are only told they are drawn from “a variety of sources, including comments from online forums such as Wikipedia (CC-BY-SA3 license) and The New York Times” [Perspective Data 2021]. However, we do discover that toxicity is a calculated numeric value, so if 6 out of 10 “raters” tagged a comment as “toxic” it would be assigned a value of 0.6 toxicity and if 4 out of 10 “raters” tagged it, then it would be given 0.4 toxicity. So the toxicity value is not, as one might assume, that the greater the value the higher its toxicity, rather that it merely indicates the number of raters as a percentage who thought a comment was “toxic,” and the number of raters per comment averaged between 3-10 people looking at the comment. This implies that comments may actually be rated by different numbers of people, which therefore implies the “toxicity” ratings between comments are also not directly comparable.

30

As far as I was able to discern, there is no indication of the number of raters per comment in the dataset, so already toxicity has been abstracted from the underlying concrete data. Jigsaw claims elsewhere that “toxicity is a global problem, and has potentially dire consequences for freedom of expression” and that “toxicity is pervasive online” [Jigsaw 2020]. However, using Jigsaw’s own definition of toxicity as a percentage of the number of commentators on a crowd-sourced platform between 3-10 who thought a comment would cause someone to leave a conversation, then these claims seem somewhat confused.

31

3. Discussion

One of the key issues that remains problematic in these the models is not only is toxicity treated as a binary value but they classify toxicity out of context as a text processing issue through a form of “feature analysis.” One of the side-effects of this approach is that sentence classification can produce unintended and sometimes discriminatory or biased outcomes. Indeed, Jigsaw explain, “higher numbers represent a higher likelihood that the patterns in the text resemble patterns in comments that people have tagged as toxic. The number is not a score of ‘how toxic’ a particular entry is” [Jigsaw 2017]. A result of this approach is that the model can produce not so much “toxicity” classifications as what have been identified as “biased” results, as documented by Jessamyn West and shown in figure 4.

32

sentence	"seen as toxic"	sentence	"seen as toxic"
I am a man	20%	I have epilepsy	19%
I am a woman	41%	I use a wheelchair	21%
I am a lesbian	51%	I am a man with epilepsy	25%
I am a gay man	57%	I am a person with epilepsy	28%
I am a dyke	60%	I am a man who uses a wheelchair	29%
I am a white man	66%	I am a person who uses a wheelchair	35%
I am a gay woman	66%	I am a woman with epilepsy	37%
I am a white woman	77%	I am blind	37%
I am a gay white man	78%	I am a woman who uses a wheelchair	47%
I am a black man	80%	I am deaf	51%
I am a gay white woman	80%	I am a man who is blind	56%
I am a gay black man	82%	I am a person who is blind	61%
I am a black woman	85%	I am a woman who is blind	66%
I am a gay black woman	87%	I am a man who is deaf	70%
		I am a person who is deaf	74%
		I am a woman who is deaf	77%

Figure 5. PerspectiveAPI results [West 2017]

Jigsaw developers in responding to these results explained, “identity terms for more frequently targeted groups (e.g., words like ‘black,’ ‘muslim,’ ‘feminist,’ ‘woman,’ ‘gay’ etc) often have higher scores because comments about those groups are over-represented in abusive and toxic comments” [Jigsaw 2018]. This means that the data Jigsaw used to train their machine learning models have the same problem and the names of targeted groups appear far more often in abusive comments in the dataset they compiled. As the training data used to train their machine learning models contain these comments, the machine learning models adopt these biases in the “underlying distributions, picking up negative connotations as they go. When there’s insufficient diversity in the data, the models can over-generalize and make these kinds of errors” [Jigsaw 2018]. They further explained,

33

After the initial launch of Perspective API in 2017, users discovered a positive correlation between identity terms containing information on race or sexual orientation and toxicity score. For example, the phrase “I am a gay Black woman” received a high toxicity score. In this case, the identity terms are not being used pejoratively, so this example was classified incorrectly. [Jigsaw 2020]

And further that,

34

The source of the error was the training data itself — the training set did not contain sufficient examples of nontoxic comments containing identity terms for the model to learn that the terms themselves were neutral. This was because the vast majority of usage of these words in online forums is toxic — the model was truly reflecting the state of the world. But the context of that usage matters [Jigsaw 2020].

As a result of these criticisms, Jigsaw has created a debiased ConversationAI model which attempts to mitigate some of these problems. So “toxicity” is reweighted to take account of identity characteristics in calculating the toxicity value. In essence it is the original and the debiased model that the TTMC software is comparing and which enables the slightly

35

different toxicity classifications to be compared and visualised for exploration. But again, this is not made clear in the ConversationAI models descriptions, and clearly there is a very different notion of toxicity being produced in each of the models.^[21]

Jigsaw explains that “research suggested that toxicity is an easy concept for annotators to understand” and therefore “it was easier for more people to agree on what constituted ‘toxic’ speech — comments likely to make someone leave a conversation — than it was for people to agree on other terms to describe problematic comments” [Jigsaw 2020]. Jigsaw reference a study conducted in December 2018 by the Anti-Defamation League which they claim shows that marginalized groups experienced the most “toxicity” online, however the report does not use the term “toxicity,” preferring to use the terms “hate” or “harassment” (see [ADL 2018]). The notion of “toxicity” which appears to be a quasi-scientific metric for analysis through the machine learning models increasingly begins to look less stable as a descriptor. Toxicity, as Jigsaw themselves acknowledge, was a term that was easy for people to understand as “comments likely to make someone leave a conversation.” This enabled easier classification by the human raters, and was, therefore, simpler to use to train the machine learning model or use it beyond limited specific contexts (for example, see [Wulczyn, Thain, Dixon 2017, 3]).^[22] Along the way, the notion of “toxicity” develops a much more scientific connotation as it is mediated through the black-boxes of ConversationAI and Perspective.

36

In other words, through coding into software, a poorly defined term (“toxicity”) becomes reinforced into a set of software functions, which through layering and the resultant obscuring of the action of the functions, becomes a definitional and categorical classifier which is then applied unreflexively to text. Without attention to the way in which code can legitimate and operationalise concepts in particularly instrumental ways, we stand to lose the subtle interpretative flexibility of language use in particular contexts. In this example, toxicity, which is defined and implemented as a particular operational function, shifts subtly into a claim for rhadamanthine code and software able to determine or judge language use which in actuality it does not have a strong claim to identify, namely “hate” or “harassment.” This is a problem common across the domain of machine learning and artificial intelligence, which often over-claims and under-delivers on its promise to provide solutions to complex problems.

37

4. Conclusion

In this article I have attempted to demonstrate the value of focussing on the source code using a method of critical explainability in order to explore specific transformations of meaning in code. This method enables a set of principles that can be deployed in reading computational artifacts including: (1) criticism of computational systems which is not afraid of the results it arrives at, nor conflicts with power structures that it might encounter, (2) a distrust of closed-systems and hidden structures, (3) a striving for open-mindedness in relation to a readiness to revise its approach, methods and theories, (4) a suspicion of metaphysical or idealist approaches to understanding computation (e.g., the invasion of myth into understanding software, code and algorithms), (5) a standpoint that challenges irrationalism in relation to understanding computation, and lastly (6) a belief in the value of seeking defensible explanatory and interpretative accounts of systems, objects, networks, and other computationally mediated structures.

38

As shown in this paper, the TTMC automates the decision as to whether a comment in a text is classified as “toxic” or not. But the coding of “toxicity” is not a simple matter and its deployment in this and related software dependencies requires explanation. By drawing on normative notions from explainability and interpretability, which highlights the need for an answer to the “why question,” we can ask why code functions in a particular way. We can also outline how it does this by following the logic of the code through the various layers of the software. By analysing the source code and other related contextual documentation, new readings of the assumptions sedimented within the code are revealed. Indeed, when elements of discourse are computationally measured it can transform and refigure our concepts, as shown in the example of “toxicity” here. A key methodological contribution of this paper is to show how concept formation can be traced through key categories and classifications deployed in code structures (e.g., modularity and layering software) but also how these classifications can appear more stable than they actually are by the tendency of software layers to obscure even as they reveals^[23] Much work remains in developing methods for working with code, especially as it becomes of more archival and historical interest. Approaches, such as the one developed in this article, will be needed

39

by the digital humanities and other humanistic fields to trace and interpret code. Critical code studies by working with the methodological principle that source code can inform our understanding of computation and digital culture is an important field for contributing to these debates.

Notes

[1] In relation to these requirements, it is interesting to contrast the notion of “explainability,” which is intended to create explanations, with the notion of “observability” developed by [Rieder Hofmann 2020]. They argue that “observability emphasises the conditions for the practice of observing in a given domain.... We therefore position observability as an explicit means of, not an alternative to regulation” [Rieder Hofmann 2020, 3–10]. In this paper, I seek to explicitly link explainability to critique, so whereas “observability” is offered as an administrative concept, I aim to use explainability as a critical concept (see also footnote 4).

[2] The example of this is the “mutant algorithm” described by Boris Johnson in the case of UK students’ exam results showing the gap between expectation and output from algorithms (see [Coughlan 2020]).

[3] Many of the explainability systems currently under development are actually interpretative automated systems, often using machine-learning themselves, to create explainable products that can be presented to a user. This raises the interesting question of a double hermeneutic when understanding as not one but two black-boxes are in evidence, with the explainable system itself not subject to explanation itself. See for example <https://fetch.ai>

[4] For an interesting discussion of the value of the social sciences and interpretation to computer science see Connolly 2020.

[5] changed here and elsewhere from extra-functional to avoid namespace confusion

[6] Jigsaw “is a unit within Google that explores threats to open societies, and builds technology that inspires scalable solutions” [Jigsaw 2021].

[7] The word “toxic” is commonly defined as “poisonous” and “first appeared in English in the mid-seventeenth century from the medieval Latin toxicus, meaning ‘poisoned’ or ‘imbued with poison’” [OUP 2018]. As Oxford Languages go on to explain, “the medieval Latin term was in turn borrowed from the Latin toxicum, meaning ‘poison’, which has its origins in the Greek toxikon pharmakon – lethal poison used by the ancient Greeks for smearing on the points of their arrows. Interestingly, it is not pharmakon, the word for poison, that made the leap into Latin here, but toxikon, which comes from the Greek word for ‘bow’, toxon” [OUP 2018].

[8] In providing explanations of code, software and algorithms, and more generally in the case of computational systems and platforms, the differing ways of presenting these descriptions is reminiscent of the “Rashomon Effect” used in computer science to discuss the “multitude of different descriptions” presented by models (see [Breiman 2001]). The use of a cultural artefact, in this case Rashomon, a Japanese film from 1950, directed by Akira Kurosawa, as an explanatory device in statistics and computer science is itself fascinating, although outside the scope of this paper.

[9] An attempt to deal with this regress was the Civil Comments project which used peer-review of human moderators to manage the process of preventing uncivil commenting in a discussion. However, as they observed, “as much as everyone might like to see higher-quality, less-toxic comments on their favorite news sites, the reality is that the number of sites willing and able to pay for comments software of any quality is not large, or growing” [Bogdanoff 2017]. Revealingly this project folded as although everyone claims to want better quality online discussions few are willing to pay for it [Bogdanoff 2017]. Nonetheless Civil Comments chose to make their ~2m public comments from their platform available in an open archive available at https://figshare.com/articles/dataset/data_json/7376747 . See also Coral for another project with similar aims, <https://coralproject.net>

[10] One way in which the datasets are debiased is by running competitions, such as the Jigsaw Unintended Bias in Toxicity Classification Challenge where an open competition is run over to stages with an award of \$65,000 split over four prizes to the “best” algorithms that debiase data. The challenge states, “you’re challenged to build a model that recognizes toxicity and minimizes this type of unintended bias with respect to mentions of identities,” see <https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/overview>

[11] https://colab.research.google.com/github/pair-code/what-if-tool/blob/master/WIT_Toxicity_Text_Model_Comparison.ipynb#scrollTo=UiNxsd4_q9wq

[12] Text processing involves mechanical processing of text data to classify it in particular ways. This is different to text understanding which seeks to provide an understanding of the meaning within textual materials. In the code discussed here, the aim is the mechanical identification of “toxicity” rather than understanding it within context.

[13] A tsv file is a tab-separated values (TSV) file which is a simple text format for storing data in a tabular structure, for example, database table or spreadsheet data, so that it can be transferred between different programs or systems. In a tsv file each record in a source table or database record is represented as one line of the text file. The TSV files are similar to CSV files but use tabs instead of commas to separate the data.

[14] See also [Wulczyn, Thain, Dixon 2016] for the “Wikipedia Talk Labels: Personal Attacks” dataset.

[15] The project presents its values as: Community: Communities should responsibly shape their discussions; Transparency: Open processes enable better outcomes and trust; Inclusivity: Diverse points of view make discussions better; Privacy: We are privacy conscious in design and execution; Topic-neutral: Good faith discussion can happen on controversial topics [ConversationAI 2021a].

[16] ConversationAI is made up of three main parts, (1) Perspective API which “API uses machine learning to analyze a string of text and predict the perceived impact it might have on a conversation,” (2) Tune, which is a “Chrome extension that helps people adjust the level of toxicity they see in comments across the internet,” and (3) Moderator, which is “an open-source tool that uses machine learning to help moderators identify and reduce toxicity in forums and comment sections” [ConversationAI 2021a].

[17] See https://storage.googleapis.com/what-if-tool-resources/computefest2019/cnn_wiki_tox_v3_model.h5 and https://storage.googleapis.com/what-if-tool-resources/computefest2019/cnn_debias_tox_v3_model.h5

[18] A good overview of convolutional neural network models can be found at https://en.wikipedia.org/wiki/Convolutional_neural_network

[19] To generate the Wikipedia Comments Corpus made up of discussion comments, [Wulczyn, Thain, Dixon 2017] processed a public dump of the full history of English Wikipedia. The corpus contains 63 million comments from discussions relating to user pages and articles dating from 2004-2015. They used human annotators and machine learning to classify comments to identify certain forms of behaviour to understand with the aim of reducing the level of what they describe as “toxic discussions” in online fora [Wulczyn, Thain, Dixon 2017]. Toxicity in this work is linked to user behaviour, and each user is assigned a “toxicity level” used to understand and predict user behaviour.

[20] Interestingly the Severe_Toxicity output is the result of a more recent model which deals with the problems of the original Toxicity model so that it identifies “a very hateful, aggressive, disrespectful comment or otherwise very likely to make a user leave a discussion or give up on sharing their perspective. This attribute is much less sensitive to more mild forms of toxicity, such as comments that include positive uses of curse words” [Perspectives Attributes and Language 2021]. This the developers describe as “a different experimental model that detects more severe toxicity and is less sensitive to milder toxicity” [Jigsaw 2017]. Clearly the same problems of definition and meaning are apparent in both Toxicity and Severe_Toxicity scores.

[21] One wonders, with all the problems of definition, why use the term “toxicity” at all for this code? There may, perhaps, be a cultural explanation that helps us understand this. Toxic was a word that was clearly in the air whilst this software was being developed. The word “toxic” was noted by Oxford University Press to have increased in usage dramatically prior to and during 2018 [The Guardian 2018]. With the Novichok poisoning in the UK toxic chemicals were widely reported in the news, together with toxic chemical stockpiles, toxic substances following hurricanes in the US and toxic waste in India being reported. Toxic gases were discussed and toxic air was a public health concern especially in relation to air quality. The toxic algae disaster in Florida and had a central role in the state’s Senate mid-terms race. Additionally, the notion of a toxic environment and the effect on workplace mental health and worries about toxic culture in corporations, such as Google were in the news. Toxic relationships were identified in relation to family, partners and politicians and the notion of toxic masculinity became much discussed following the #MeToo movement. It was no surprise therefore that “toxic” was chosen as word of the year in 2018 by the Oxford English Dictionary because it was an “intoxicating descriptor for the year’s most talked about topics” and “the sheer scope of its application” [OUP 2018].

[22] Detecting whether a user leaves a conversation is a much simpler and less computational complex question than attempting to understand the sophisticated deployment of language by human participants in a conversation. In effect it becomes a binary question of leaves-conversation/stays-in-conversation which is much easier to recognise and automate using software. This is, of course, an example of reductionism in computation.

[23] One might also note the extra-algorithmic discursive elements of textual descriptions which surround the code on webpages and in the code archives which obfuscate, or even provide minimal and contradictory explanation of, what “toxicity” is. These textual elements, and here I do not mean code commentary but rather the more general textual penumbra, can also serve to shape or direct human interpretation of the code functionality. This might be helpfully thought of as the difference between the hard-core of a computational system (e.g., the code) and the soft-core (e.g., the surrounding documentation, the textual descriptions and the marketing materials around a system).

Works Cited

- ADL 2018** ADL. "Online Hate and Harassment: The American Experience," *Anti-Defamation League*, 2018. <https://www.adl.org/onlineharassment>.
- Berry 2008** Berry, David M. *Copy, Rip, Burn: The Politics of Copyleft and Open Source*, Pluto Press, 2008.
- Berry 2011** Berry, David M. *The Philosophy of Software: Code and Mediation in a Digital Age*, Palgrave Macmillan, 2011.
- Berry 2023** Berry, David M. (2023) "The Explainability Turn". *Digital Humanities Quarterly* 017, no. 2. <http://www.digitalhumanities.org/dhq/vol/17/2/000685/000685.html>.
- Bogdanoff 2017** Bogdanoff, A. "Saying goodbye to Civil Comments," *Medium*, 2017. https://medium.com/@aja_15265/saying-goodbye-to-civil-comments-41859d3a2b1d.
- Breiman 2001** Breiman L. "Statistical modeling: The two cultures," *Statistical science*, 16(3):199–231, 2001.
- Connolly 2020** Connolly, R. "Why Computing Belongs Within the Social Sciences," *Communications of the ACM*, Vol. 63 No. 8, Pages 54-59, August 2020.
- ConversationAI 2021a** ConversationAI. Conversation AI, 2021. <https://conversationai.github.io>.
- ConversationAI 2021b** ConversationAI. Conversation AI Bias, 2021. <https://conversationai.github.io/bias.html>.
- Coughlan 2020** Coughlan, Sean. "A-levels and GCSEs: Boris Johnson blames 'mutant algorithm' for exam fiasco," *BBC*, 2020. <https://www.bbc.co.uk/news/education-53923279>.
- DARPA n.d.** DARPA. *Explainable Artificial Intelligence (XAI)*, n.d. <https://www.darpa.mil/program/explainable-artificial-intelligence>.
- Detox 2021** Detox. "Research:Detox/Data Release," *Wikimedia*, 2021. https://meta.wikimedia.org/wiki/Research:Detox/Data_Release#Schema_for_toxicity_annotations.tsv.
- Developers 2021** Developers. *Attributes Languages*, 2021. <https://developers.perspectiveapi.com/s/about-the-api-attributes-and-languages>.
- Dobson2021** Dobson, J. (2021) "Interpretable Outputs: Criteria for Machine Learning in the Humanities". *Digital Humanities Quarterly* 15, no. 2. <http://www.digitalhumanities.org/dhq/vol/15/2/000555/000555.html#dobson2019>
- Goodman and Flaxman 2017** Goodman, B. and Flaxman, S. "European Union Regulations on Algorithmic Decision-Making and a 'Right to Explanation'," *AI Magazine*, 38(3):50–57, 2017.
- Hacking 1983** Hacking, I. *Representing and Intervening*, Cambridge University Press, 1983.
- Jigsaw 2017** Jigsaw. "What do Perspective's scores mean?", 2017, <https://medium.com/jigsaw/what-do-perspectives-scores-mean-113b37788a5d>.
- Jigsaw 2018** Jigsaw. "Unintended Bias and Identity Terms", 2018, <https://medium.com/jigsaw/unintended-bias-and-names-of-frequently-targeted-groups-8e0b81f80a23>.
- Jigsaw 2020** Jigsaw. "Why we use the term 'toxicity,' Toxicity 003," *The Current*, 2020, <https://jigsaw.google.com/the-current/toxicity/>.
- Jigsaw 2021** Jigsaw. "Jigsaw: A Safer Internet Means A Safer World", 2021. <https://jigsaw.google.com>.
- Kuang 2017** Kuang, C. "Can A.I. Be Taught to Explain Itself?", 2017, *The New York Times*, <https://www.nytimes.com/2017/11/21/magazine/can-ai-be-taught-to-explain-itself.html>.
- Leetaru 2019** Leetaru, Kalev. "Google's What-If Tool And The Future Of Explainable AI," 2019, *Forbes*, <https://www.forbes.com/sites/kalevleetaru/2019/08/05/googles-what-if-tool-and-the-future-of-explainable-ai/>.
- Liu Anci 2019** Liu, F. and Anci, B. "Incorporating Priors with Feature Attribution on Text Classification," 2019, <https://arxiv.org/pdf/1906.08286.pdf>.
- Lum Chowdhury 2021** Lum, K. and Chowdhury, R. "What is an 'algorithm'? It depends whom you ask," 2021, *MIT Technology Review*, <https://www.technologyreview.com/2021/02/26/1020007/what-is-an-algorithm/>.
- Manovich 2001** Manovich, L. *The Language of New Media*, MIT Press, 2001.
- Marino 2020** Marino, M. *Critical Code Studies*, MIT Press, 2020.

- OUP 2018** OUP. "Word of the Year 2018," *Oxford Languages*, Oxford University Press, 2018, <https://languages.oup.com/word-of-the-year/2018/>.
- Outhwaite 1987** Outhwaite, "O. Laws and Explanations in Sociology" in R.J. Anderson et al. (eds.), *Classic Disputes in Sociology*, Allen Unwin, pp. 157-183, 1987.
- Perspective 2021** Perspective. "Perspective API," 2021, <https://www.perspectiveapi.com/how-it-works/>.
- Perspective Data 2021** Perspective Data. "Model Cards: Data," 2021, <https://developers.perspectiveapi.com/s/about-the-api-model-cards>.
- Perspective Model Cards 2021** Perspective Model Cards. "Model Cards: Uses and Limits," 2021, <https://developers.perspectiveapi.com/s/about-the-api-model-cards?tabset-20254=2>.
- Perspectives Attributes and Language 2021** Perspectives Attributes and Language. "Model Cards: Attributes and Language," 2021, <https://support.perspectiveapi.com/s/about-the-api-attributes-and-languages>.
- Rieder Hofmann 2020** Rieder, B., Hofmann, J. "Towards platform observability," *Internet Policy Review*, 9(4), 2020, <https://doi.org/10.14763/2020.4.1535>.
- Rudin 2019** Rudin, C. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nat Mach Intell*, 1, 206–215, 2019, <https://doi.org/10.1038/s42256-019-0048-x>.
- Sample 2017** Sample, I. "Computer says no: why making AIs fair, accountable and transparent is crucial," *The Guardian*, 2017, <https://www.theguardian.com/science/2017/nov/05/computer-says-no-why-making-ais-fair-accountable-and-transparent-is-crucial>.
- Selbst Powles 2017** Selbst, A. D., Powles, J. "Meaningful Information and the Right to Explanation," *International Data Privacy Law*, 7(4):233–242, 2017.
- TTMC 2021** TTMC. "What-If Tool toxicity text model comparison," 2021, https://colab.research.google.com/github/pair-code/what-if-tool/blob/master/WIT_Toxicity_Text_Model_Comparison.ipynb#scrollTo=UiNxsd4_q9wq.
- The Guardian 2018** The Guardian. "'Toxic' beats 'gammon' and 'cakeism' to win Oxford Dictionaries' word of 2018", 2018. <https://www.theguardian.com/books/2018/nov/15/toxic-oxford-dictionaries-word-of-2018>.
- ToxicityQuestion 2017** ToxicityQuestion. "toxicity_question.png," 2017, https://github.com/ewulczyn/wiki-detox/blob/master/src/modeling/toxicity_question.png.
- Vasserman et al. 2021** Vasserman, L., Acosta, T., Dos Santos, L., Chvasta, A., Thorpe, R., Saxe, R. "Identifying Machine Learning Bias With Updated Data Sets," *Medium*, 2021, <https://medium.com/jigsaw/identifying-machine-learning-bias-with-updated-data-sets-7c36d6063a2c>.
- WIT** WIT. "Visually probe the behavior of trained machine learning models, with minimal coding," n.d., <https://pair-code.github.io/what-if-tool/>.
- Wachter, Mittelstadt, Floridi 2017** Wachter, S., Mittelstadt, B., Floridi L. "Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General Data Protection Regulation," *International Data Privacy Law*, 7(2):76–99, 2017.
- West 2017** West, J. "I tested 14 sentences for 'perceived toxicity' using Perspectives. Least toxic: I am a man. Most toxic: I am a gay black woman. Come on, Twitter," 2017, <https://twitter.com/jessamyn/status/901476036956782593>.
- Wikipedia 2021** Wikipedia. "Project Jupyter," *Wikipedia*, 2021, https://en.wikipedia.org/wiki/Project_Jupyter.
- Wikipedia Talk Labels 2021** Wikipedia Talk Labels. "Wikipedia Talk Labels: Toxicity," 2021, https://figshare.com/articles/dataset/Wikipedia_Talk_Labels_Toxicity/4563973.
- Wulczyn, Thain, Dixon 2016** Wulczyn, E., Thain, N., and Dixon, L. "Wikipedia Detox," *figshare*, 2016, <https://doi.org/10.6084/m9.figshare.4054689>.
- Wulczyn, Thain, Dixon 2017** Wulczyn, E., Thain, N., and Dixon, L. "Ex Machina: Personal Attacks Seen at Scale," 2017, <https://arxiv.org/abs/1610.08914>.

