Logo, company name

Description automatically generated

Data Science & Artificial Intelligence: DSI 2

XXXXXXXXX – 2021/2022

Date: 10/07/2022  
Version: 1.0

Team Equal Web

|  |  |
| --- | --- |
| Name | Student Number |
| XXXXXX XXX XXXX | XXXXXXXX |

Document Versions

|  |  |  |
| --- | --- | --- |
| Version | Date | Status |
| 0.1 | 05/14/2022 | First concepts. Made preface, introduction & started on LO 1. |
| 0.2 | 05/25/2022 | Second version, decided on model, datasets, techniques to use etc. |
| 0.3 | 06/06/2022 | Finished coding, cleaning, modelling, etc. Started writing LO 3 and 4. |
| 0.4 | 15/06/2022 | Finished LO 3 and 4. |
| 0.5 | 22/06/2022 | Started on Remainder of LO 2. |
| 0.6 | 30/06/2022 | Finished half of LO 2. |
| 0.7 | 08/07/2022 | Finished remainder of LO 2. Writing afterword, fixing references, fixing terminology, etc. |
| 1.0 | 10/07/2022 | Finalized version. Checked spelling, adjusted some texts, added page numbers, fixed figure numbers, etc. |

Preface

This portfolio concerns itself with a Data Science project, made for the Data Science & Artificial Intelligence DSI 2 course. Due to the project being done as a one-man party, more freedom was allowed in terms of topic and subject matter. For this reason, an intentionally new and challenging subject was chosen to focus on: Natural Language Processing (NLP). To these ends, the fictional organization The Equal Web was created which mainly concerns itself with performing research into digital spaces on the internet that are known for fuelling and perpetuating hate, discrimination, and prejudice.

As a new topic of study, NLP is both exciting and daunting at the same time. However, it’s simultaneously a topic in the Data Science world which has recently exploded in popularity. Due to a plethora of sources and study material being available at a beginner level, it was possible to acquire sufficient knowledge for the utilization of these types of models. The biggest hurdle to overcome was the uncertainty of whether the result would even work. While this choice was a bit of a gamble, I was determined to pull through and dedicate myself to achieving results to be satisfied with.

Overall, this project allowed for a low-level incursion into the field of NLP to spark new interest, making it something I might potentially pursue in the future.

Table of Contents

**Preface 2**

**Introduction 4**Client 5Project Description 6

**Terminology 8**

**Learning objective 1: You set up a Data Science process 11**1.1 You describe data mining activities based on choice of a basic machine learning model technique and relevant required activities 111.2 You define data mining success criteria 201.3 You add extra self-organized and/or external data sources to the Data Science process 23

**Learning objective 2: You collect and address relevant data 32**2.1 You (re-)validate data after model-generated assumptions 322.2 You integrate relevant data by merging multiple data sources 442.3 You clean data by imputing and scaling relevant data 502.4 You construct data by one-hot-encoding, defining targets & labelling relevant data 612.5 You convert data formats as prerequisite for relevant model technique(s) 69

**Learning objective 3: You perform data analysis 74**3.1 You split data into test & train sets to generate a test design 743.2 You build & train relevant model technique(s) and create predictions using the model technique(s) on test data set 773.3 You assess the model technique(s) on chosen metrics of the defined success criteria 88

**Learning Objective 4: You evaluate & deploy results of the Data Science process 92**4.3 You produce a deliverable for the customer 924.1 You evaluate and match success criteria with business objectives of the Data Science   
process 974.2 You determine next steps and set up an advisory report for follow-up 1004.4 You review the Data Science process and collect lessons learned on the  
process & product 102

**Afterword 104**

**References 105**

Introduction

This portfolio will give further introduction about the client, their wishes, and all the preparations and necessary steps that have been taken to bring the project to completion. Due to the intrinsically complex nature of Natural Language Processing, ample explanations and visual aid will be provided wherever necessary. Overall, the goal of this document is to allow for a proper understanding of the operations performed to aid The Equal Web’s ongoing battle against hate to flourish. This combatting of hate is a constant effort and preventing hate from festering inside a community is necessary. However, The Equal Web realizes that merely displacing the symptom does nothing to address any potential root causes of the problem. Daryl Davis, an African American musician put in a concerted effort in the 1980s to improve race relations by seeking out and engaging in dialogue with members of the Ku Klux Klan (Davis, 2017). He eventually befriended one of their leaders in Maryland, Roger Kelly, as well as other members, and over time convinced them to leave the Klan. Daryl Davis wished to truly understand the source of the hatred that he had personally witnessed, and the main lesson to take away from his story resides within one of his quotes:

*“The lesson learned is: ignorance breeds fear. If you don’t keep that fear in check, that fear will breed hatred. If you don’t keep that hatred in check, it will breed destruction.”*

In similar fashion to Daryl Davis’ efforts, The Equal Web wishes to better understand the causes and underlying dynamics of places and organizations wherein hatred is a natural part of life, and to discover how best to resolve these issues. They argue that without this proper understanding, their modus operandi becomes a paradoxical one, as it would require to be intolerant of intolerance. Persecution tends to only strengthen those that feel unfairly treated, and it breeds a bitterness that results in lashing out against the world. Truly understanding is what The Equal Web wishes to achieve in the future, and they realized that their current perception does not suffice.

**Client**

The Equal Web is an organization that strives to make the digital space a hospitable environment for every single individual, regardless of gender, religion, upbringing, social standing, or belief. They are a moderately large entity, with several hundred employees. Due to its ties with the digital world, the organization had a heavy focused on technology, research, and development.

Since its inception, the internet has always been a Wild West of possibilities and utilization, becoming the source of equally wonderful and dreadful events. One of the internet’s most popular usages, without a doubt, relates to discussion. Many places of discussion exist, such as forums, wherein anyone with an internet connection is able to join and contribute to the topics of discussion that are prevalent. It’s also no secret that certain individuals derive joy from disrupting said discussion and sowing discord wherever possible. Factors such as real-world distance, anonymity, and lack of consequences only embolden this behaviour. Mild pranks aside, this allows for dangerous streams of thought to be easily shared as well. Modern efforts to moderate these spaces through proper monitoring, automatic filtering, and the usage of bots, have contributed greatly to ensuring that people are protected from bigotry and bad actors. However, just like a game of cat and mouse, this goal is a continuous effort and requires its participants to be up to date with the ever-changing rules and conditions of the game.

The Equal Web mainly concerns itself with providing advice related to moderation of digital space, the creation of advanced and customized filters, the creation of infrastructure to accommodate new services that wish to have an early start with providing safety and comfort, and performing research into new techniques and technologies that could aid themselves as well as others. Due to the nature of this document’s contents, the project has close relations with The Equal Web’s research & development team.

**Project Description**

While noble in spirit, a root cause is not easily discovered or understood, especially since The Equal Web concerns itself with digital spaces that are very fragmentized and whose participants originate from all over the globe. Regardless, adequate research will pave the way for the future through the creation of necessary fundamentals. The direct issue at hand is that this manner of research would require exactly that which has already largely been shunned from the internet. This would require a community that operates on rules that are the inverse of the usual, modern community: a space with hardly any moderation, filtering, and restrictive rulesets. However, these spaces continue to exist till today, and they are not exactly hidden either. As this project will serve as a humble beginning to a vast journey of discovery, grandiose and ground-breaking discoveries are not expected. Instead, it is expected for the project to produce that which Data Science is exactly intended for: statistical insights that will allow for a better understanding of the situation. For this project, a community wherein hatred is perpetuated needs to be discovered, proper machine learning techniques need to be chosen, and all findings need to be visualized and summarized in an advisory report.

The field that will be relevant to this research paper is of course the field of Data Science. By following a methodology such as CRISP-DM (Cross Industry Standard Process for Data Mining) from figure 1, the steps that are usually present in a Data Science-process can be followed in an organized and methodical fashion. This choice is backed up by how Data Science aims to generate insights and visual aid based on large quantities of data in a scientific manner through complex mathematics and statistical models. Considering the problem at hand, its purposes and The Equal Web’s intentions complement each other.



*Figure 1: The CRISP-DM cycle (Data Science Process Alliance, 2022)*

It’s also necessary to find a proper community to observe. It’s required that moderation does not get in the way of harvesting data. The rules should be lax, and open to the sharing of ideas and opinions that go against the grain. There also needs to be enough user traffic for the acquisition of enough data to fuel any statistical models. Whereas many online forums operate on a much cleaner basis nowadays, some places that fulfil the mentioned requirements have withstood the test of time and remain active to this day. One such place is the website known as 4chan. This website is an anonymous English-language imageboard website, first launched in 2003. Within its many boards (which are focused on a singular overarching topic, such as cooking or fitness), all kinds of sub-communities are found.

With more than 22 million unique monthly visitors as of 2022, the website’s more sinister boards have been the source of many pranks, scandals, and controversy. One topic of discussion that always tends to have a volatile nature is politics. As such, this research will rely on the /pol/ board (politically incorrect) for its data acquisition. Due to 4chan’s casual moderation and inherent anonymity, even the worst opinions and calls to violence are allowed to subsist. Within a board, a user is capable of starting a discussion by creating what’s known as a thread: a topic of discussion that exists as its own entity within the board. Other users can then reply to this thread and post images. Threads do not last forever, however, as either inactivity or a reply limit will cause them to eventually be removed. As new threads are made, older ones must make place for them. These features make this website different from traditional forums: anonymity and transience. However, archiving services do exist that temporarily save deleted threads, which will be relied upon for data acquisition.

All this gleamed information can be distilled to formalize an overarching business goal for this project. This goal is formulated as such:

*“Utilize Data Science and machine learning to measure and visualize the prevalence of hate within the /pol/ community”*

A business goal is difficult to achieve in one sitting, and it’s usually not a single concrete task that can be worked on. Business success criteria can be created to allow for a more structured approach. To properly validate whether the final products of this project have worth, these success criteria are required to keep in mind as parameters of success. For this project, the success criteria are as follows:

* Distinctions are made between hateful and non-hateful speech
* The volume of each speech type is categorized and visualized

Terminology

It’s possible for many terms or abbreviations within this paper to be of foreign origin as they are related to fringe concepts or sections of the internet. To ensure legibility, these terms are clarified in this section of the document. In addition, they will be divided between 4chan-related terms and terms related to machine learning among tables 1 and 2.

|  |  |
| --- | --- |
| Term | Definition |
| 4chan | An imageboard website created in 2003 by Christopher Poole, also known as “moot”. |
| Board | A subsection of the 4chan website, dedicated to specific topics such as cooking or fitness. |
| /pol/ | The politically incorrect board of 4chan, dedicated to politics. |
| Thread | A topic any user may create in a board, allowing other users to reply and join the discussion. |
| OP | The Original Poster, otherwise known as the creator of the thread. |
| ID | A unique, alphanumeric string available on some boards, allowing for the identification of the same individual posting multiple times. IDs do not carry over between threads. |
| Sticky | A thread that has been pinned to the start of the board by moderators, usually for the purpose of showing board rules. |
| Greentext | A way of typing by prefixing a line of text with the right-facing arrow symbol (“>”), used for quoting or storytelling purposes. |
| Tripcode | A unique hash users can use to be identifiable across threads and boards which is very infrequently used and looked down upon by other users. |
| Memeflag | Using a non-geographical flag to denote an alignment. This is also looked down upon by other users as it’s believed this is done to hide the country of origin. |
| Get | A post with a number consisting of repeated digits or special patterns of digits, e.g., 1234567 or 2222222. |
| Dubs, trips, quads, etc. | A get ending in a certain number of repeating digits. Dubs refers to two repeating digits, trips to three, quads to four, and so on. |
| NSFW | Not Safe For Work |

*Table 1: 4chan-related terminology*

|  |  |
| --- | --- |
| Term | Definition |
| NLP | Natural Language Processing, a field of linguistics and machine learning focused on understanding everything related to human language. |
| BERT | Bidirectional Encoder Representations from Transformers, a language representation model used for NLP. Servers as an architecture for other BERT-based models. |
| Transformer | A neural network that learns context and thus meaning by tracking relationships in sequential data. |
| Architecture | The skeleton (such as BERT) of a model – the definition of each layer and each operation that happen within the model. |
| Checkpoint | The weights that will be loaded in each architecture. |
| Model | An umbrella term that isn’t as precise as “architecture” or “checkpoint”: it can mean both. |
| Sequence | A string containing words and punctuation, can contain any number of words or sentences. |
| Tokenization | Component that is capable of transforming text into numerical data, based on a given dictionary of words. |
| Attention mask | A binary representation of which Input IDS need to be ignored in each input sequence. |
| Padding | Lengthening smaller sequences in a batch with tokens representing 0 to ensure the batch is of rectangular shape. |
| Input ID | Numerical representation of a token, created by the tokenizer. |
| Token | A character, word, or sub-word in a sequence. |
| Tensor | A generalization of vectors and matrices and is easily understood as a multidimensional array. |
| Logit | Raw, unnormalized scores output by the last layer of the model. |
| SoftMax | Mathematical function that takes the input of a vector of real numbers and normalizes it into a probability distribution. Used to convert logits into probabilities. |
| Scaling | Changing the value range of numerical data to make distance-based algorithms more accurate when used on said data. |
| Discretization | Converting continuous data attribute values into a finite set of intervals and associating with each interval some specific data value. |
| One-hot-encoding | A method of converting data into numerical values as preparation for use by a machine learning algorithm. |
| Epoch | The number of passes of the entire training dataset the machine learning algorithm has completed during the fine-tuning of a checkpoint. |

|  |  |
| --- | --- |
| Weight decay | A regularization technique that causes the weights during training to exponentially decay to zero by adding a small penalty over time. This reduces the impact of certain inputs over time, and it is also used to prevent overfitting. |
| Warmup steps | How many updates with a low learning rate are performed before the training uses its default learning rate. |
| Learning rate | A parameter that determines how much an updating step influences the current value of the weights. The weights, in turn, determine how much influence certain input has over the output. |
| Metric | Assessment technique used to measure the quality of the statistical or machine learning model. |
| Accuracy | The proportion of correct predictions among the total number of cases processed. It can be computed with: Accuracy = (TP + TN) / (TP + TN + FP + FN) |
| F1 | The harmonic mean of the precision and recall. It can be computed with the equation: F1 = 2 \* (precision \* recall) / (precision + recall) |
| ROC AUC | This metric computes the area under the curve (AUC) for the Receiver Operating Characteristic Curve (ROC). The return values represent how well the model used is predicting the correct classes, based on the input data. |

*Table 2: Machine learning-related terminology*

Learning Objective 1: You set up a Data Science Process

**1.1 You describe data mining activities based on choice of a basic machine learning model technique and relevant activities**

As this project concerns itself with the exploration of an online environment whose main produce is discussion, it’s natural that said discussion will be the focus of any research that will be performed. However, computers do not operate on words and languages like humans do. Simple sentences such as “I am hungry” or “I would like an apple” are easily interpreted by the human conscience, and it’s understood that these sentences may be related to one another. The interpretation of human language is difficult to pull off as it needs to be processed in such a way that a model is capable of learning from it, and it’s important to carefully think of how this processing can be done due. With the advent of machine learning, one specific field has recently come to surface that focuses specifically on the computerized interpretation and handling of language: Natural Language Processing (NLP). According to Hugging Face (n.d.), NLP is a field of linguistics and machine learning focused on understanding everything related to human language. Just like the applications of a language, the field itself is very broad. Several common examples of NLP tasks, with examples, are:

* **Classifying whole sentences:** Getting the sentiment of a review, detecting if an email is spam, determining if a sentence is grammatically correct or whether two sentences are logically related or not
* **Classifying each word in a sentence:** Identifying the grammatical components of a sentence (noun, verb, adjective), or the named entities (person, location, organization)
* **Generating text content:** Completing a prompt with auto-generated text, filling in the blanks in a text with masked words
* **Extracting an answer from a text:** Given a question and a context, extracting the answer to the question based on the information provided in the context
* **Generating a new sentence from an input text:** Translating a text into another language, summarizing a text

Even through a few examples, it’s clear that the multi-faceted toolbox of NLP contains techniques that might be useful for this project as well. As this project concerns itself with classifying hate, the classification of sentences and sentiment analysis sound appealing for attaining this project’s goals. The website of Hugging Face is also one that will return many times in this document. Hugging Face has undergone an explosive growth that elevated it to a status of “the GitHub for Natural Language Processing”. The website features many tutorials, documentations, and it houses a plethora of models, datasets, and introductory courses to follow.

After more research, it has become known that while the field is relatively in its infancy compared to other applications of Data Science, large strides of improvement have been made in recent years. In 2017, Google had developed a new neural network architecture known as Transformers which, according to Uszkoreit (2017), vastly outperforms both recurrent and convolutional models on academic English to German and English to French translation benchmarks, as visible in figures 2 and 3.

Translation of language can be very difficult to do due to grammatical differences, sentence structures, gendered language, etc.

Chart, bar chart

Description automatically generated

*Figure 2: BLEU scores (higher is better) of single models on the standard WMT newstest2014 English to German translation benchmark.*

Chart, bar chart

Description automatically generated

*Figure 3: BLEU scores (higher is better) of single models on the standard WMT newstest2014 English to French translation benchmark.*

Google’s research paper on this subject, Attention Is All You Need by Vaswani et al. (2017 ), goes into more detail on the mechanisms of their Transformer architecture, which will be used later in this document. The Transformer architecture has since been expanded upon for utilization in all sorts of Natural Language Processing tasks. Other examples of Transformer architectures that have since been developed are GPT in June 2018, BERT in October 2018, GPT-2 in February 2019, DistilBERT in October 2019, BART and T5 in October 2019, and GPT-3 in May 2020. This list is far from complete, and mainly shows a few different types of Transformer models.

Despite any differences, all these models have been trained as language models. This means that they have been trained on large quantities of raw text in a self-supervised manner. Self-supervised learning is a type of training in which the objective is automatically computed from the inputs of the model. This way, the model develops a statistical understanding of the language that it has been trained on, but it is not useful nor practical for specific tasks. For this reason, these pretrained models then undergo a process called Transfer Learning. During this process, the model is fine-tuned in a supervised way on a given task by using human-annotated labels. More information on how the Transformer models work will become relevant later in this document.

The website of Hugging Face is also dedicated to these Transformer architectures and houses many models that are based off them. At this point in time of the research, several key points have become clear that will steer any decision-making into the correct direction. These key points are:

* Transformers are the latest, trendiest, and best Natural Language Processing technology
* The relevant Natural Language Processing task for this project is Text Classification
* Transformer models are readily available on Hugging Face

However, there are several caveats to these Transformer models. The first issue is that training a model is very costly in terms of time and resources. This also translates to environmental impact, with the creation and training of the larger models emitting more CO2 than the entire lifetime of a single American car, including fuel. Comparisons with CO2 emissions of other activities are visible in figure 4.

A picture containing graphical user interface

Description automatically generated

*Figure 4: Comparison of CO2 emissions between Transformer models and other common activities*

Additionally, creating one from scratch is an extremely complex task that would require terabytes of data, millions of parameters, and weeks to months of time to train. Naturally, this is way beyond the scope of this Data Science project. As development continued, so has the growth in size of these models, as visible in figure 5.

Chart, timeline, scatter chart

Description automatically generated

*Figure 5: Transformer model sizes and number of parameters (in millions)*

Due to these size and time constraints, it is only feasible to use pre-trained Transformer models found in spaces such as Hugging Face. At first glance, it seems that this would sabotage the entire process of training a model which is required for the project, but this is not the case. It is possible to fine-tune a model by allowing it to go through a manual training process. This is also possible to perform by using your own, custom data.

Another issue comes up related to the supply of this data for the training process. As the main task for this project concerns itself with the classification of text as hateful or not, it will be necessary to discover a model that can do this in a satisfactory way. However, the labelling of data is based on human judgment, meaning that whoever created the model had a certain idea of what is hate speech or not. Were new data to be supplied to fine-tune the model, the labels and rationale would need to be the same. The manual labelling of custom data for this end would thus be very arduous, time-consuming, and potentially impossible due to not knowing the model creator’s rationale. For this reason, it’s important to first scout for NLP models and datasets that are compatible with each other.

Before talking more about models, it’s important to clarify several terms that have been used so far and will be used more in the upcoming sections of this document. These terms are:

* **Architecture:** This is the skeleton of the model: the definition of each layer and each operation that happens within the model.
* **Checkpoints:** These are the weights that will be loaded in each architecture.
* **Model:** This is an umbrella term that isn’t as precise as “architecture” or “checkpoint”. It can mean both.

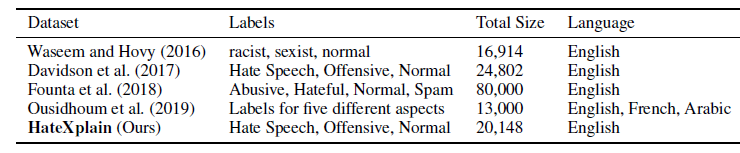
For example, BERT would be an architecture, while bert-base-uncased is a checkpoint: a variation based on the BERT architecture. The term model is thus interchangeable, but the more specific terms will be used from now on to reduce ambiguity. With these potential issues and distinctions understood, research has been performed into what Hugging Face has to offer on its website. With a quick search for the term “hate”, several models dedicated to the classification of hate speech already came to light. Examples of these are:

* Hate-speech-CNERG/bert-base-uncased-hatexplain
* beomi/beep-KcELECTRA-base-hate
* pysentimiento/robertuito-hate-speech

The second and third choices were quickly disqualified due to not satisfying the constraints that were named earlier. The robertuito-hate-speech checkpoint, while looking very promising due to its multi-classifier model, is more complex than bargained for, and is also capable of detecting emotion, hate speech, irony, and sentiment. It’s also unclear what data it has been trained on, making it difficult to secure additional data that would be in identical format.

The beep-KcELECTRA-base-hate checkpoint is completely lacking in documentation, making it unknown how the model was created and how data can be secured to further fine-tune the model.

The bert-base-uncased-hatexplain checkpoint is very promising, however. The model is capable of classifying text as normal, offensive, or hate speech. In addition, the model links to a GitHub repository, which then links to the research paper that was published alongside this checkpoint. According to Mathew et al. (2021), data has been collected from other sources where prior research into hate speech has been conducted. These sources are visible in figure 6.



*Figure 6: Data sampling sources of the HateXplain Transformer checkpoint*

As visible within figure 5, there exists another study that used the exact same labels as the HateXplain checkpoint. This data originates from Davidson et al. (2017) and is available for usage online. The existence and availability of this data means that HateXplain is a viable candidate for use, as well as fine-tuning. With all this information, the availability of resources, and knowledge of model requirements that have been gathered so far, a conclusion was possible to make. Thus, the HateXplain model has been chosen to use for this Data Science project.

Additional research was performed into the steps that will need to be performed to utilize these Transformer models from Hugging Face. It can be as simple as importing the pipeline function from Hugging Face’s Transformers library, which hides all the work behind the scenes. However, it’s also possible to perform all the necessary steps manually, which is the intention for this project as well. Figure 7 shows the processes that will need to be completed.

Diagram

Description automatically generated

*Figure 7: The Transformer model pipeline processes*

As these Transformer models are language models, they will be given a plethora of textual input, usually in the form of strings. This is the raw text visible in figure 7, with the example string of “This course is amazing.” However, computers do not function based on regular human language, and operate on numerical values instead. For this reason, the input text will need to go through a process referred to as tokenization. Tokenization is the conversion of raw text input to integer values, referred to as tokens. There are three main ways of performing tokenization, which are:

* Word-based tokenization
* Character-based tokenization
* Subword-based tokenization

The first implementation of word-based tokenization is simple in concept, and relates to the splitting up of strings based on words, punctuation, whitespace, etc. An example is given in figure 8.

Table

Description automatically generated

*Figure 8: Word-based tokenization*

This type of tokenization looks good on paper, but it has several flaws. First, to complete cover a language, a huge number of tokens will be necessary. For example, there are over 500,000 words in the English language, so to build a map from each word to an input ID we’d need to keep track of that many token IDs. In addition, words like “cat” and “cats” will be represented differently, and the model will have no way of knowing that they are related. In addition, a special token known as the unknown token is needed for words that are not present in the vocabulary given to the model. This token is usually represented by [UNK]. It’s generally not a good sign if your model is producing a lot of unknown tokens.

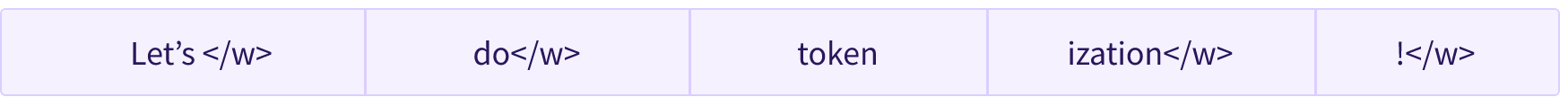
One way of reducing these unknown tokens is by implementing character-based tokenization instead. How text is split by using this method is visible in figure 9.



*Figure 9: Character-based tokenization*

This approach isn’t perfect either due to how it’s based on characters instead of words. Intuitively, each character does not have a lot of meaning by itself, compared to words. This functions differently in languages such as Chinese where each character carries a lot of meaning. This method also means an inherently massive number of tokens for the model to process, compared to the situation where each word is only one token.

There is a remedy to the issues posed by both tokenization methods, which is the third category mentioned: subword-based tokenization. Subword-based tokenization algorithms rely on the principle that frequently used words should not be split into smaller subwords, but rare words should be decomposed into meaningful subwords. An example of this method is visible in figure 10.



*Figure 10: Subword-based tokenization*

This method avoids large quantities of tokens yet still considers variations of words. Different words such as “runs”, “running”, “runner”, etc., would all be split up into “run”, with the suffixes being granted their own token ID. This way, common suffixes such as -ing, or -ly can be reused for other situations where similar verbiage is used. This allows the model to have relatively good coverage with small vocabularies, and close to no unknown tokens. In addition, it’s extra useful for languages such as Dutch, German, Turkish, etc. which allow for arbitrarily long and complex words by combining many subwords.

There are additional methods of tokenization as well, but these 3 are the main ones. The BERT-based checkpoints make use of sub-word tokenization. Since the chosen Hate-speech-CNERG/bert-base-uncased-hatexplain checkpoint relies on BERT as well, this method is present in its tokenization. In addition, as these models are all pre-trained, their tokenization methods can be imported as well. This is important to guarantee the tokenization happens exactly according to how it was performed for this checkpoint. This way, each token ID will retain the connection with the same word.

Figure 7 shows the results of the tokenization, referred to as Input IDS, which is: [101, 2023, 2607, 2003, 6429, 999, 102]. It’s visible that there are more tokens present than the original raw input has words. The 101 and 102 tokens are special tokens that denote the beginning and end of a sentence. The remaining tokens add up to a quantity of 5, whereas the raw input had 4 total words. It’s thus likely that “amazing” has been split up into 2 tokens using subword-based tokenization.

After the tokenization has been done, the input is given to the model, the model works its magic, and then output is given. Figure 7 shows that the output is given in logits. These are not probabilities but instead the raw, unnormalized scores output by the last layer of the model. These can be transformed into probabilities using a SoftMax function, by using either PyTorch or Tensorflow. The SoftMax function is a complicated, mathematical function that takes the input of a vector of real numbers and normalizes it into a probability distribution, based on how many probabilities there are. In figure 7, there are two probabilities which are positive and negative. Shown in the image are the resulting percentage values of each probability, together summing up to 100%.

There are a few assumptions to make about the required data for various steps during the project. Firstly, multiple datasets will be present. The Davidson et al. (2017) dataset will need to be imported, which needs to have the exact same labels and meaning as the HateXplain checkpoint. There will also need to be an adequate split between label frequency. For example, if the dataset only contains 5 sentences of hate speech, it will not be of sufficient quality to perform fine-tuning.

Data will also need to be collected from the /pol/ community somehow. There aren’t too many requirements for this data as it needs not be manually labelled. The texts do need to contain English as that is the language the HateXplain model is based on. In addition, the user comments need to be available, as well as additional data about geographical location to create visualizations that will grant insight into user distribution and hate distribution. Research has been performed and it’s known that this data can be collected from an API, which will allow for the harvesting of large quantities of text data. This is more expanded upon in a later section of the document.

Finally, the chosen model is a supervised machine learning technique due to its utilization of human-based labelling. This is required to attach meaning to connection and correlations at the point of Transfer Learning that the model previously discovered during self-supervised learning. While the model finds words that are likely to be relevant to each other during self-supervised learning, it will not know what this connection is. Meaning is only given once labels come into play, at which point the model turns into a supervised machine learning technique instead. An unsupervised NLP technique will not contribute to this project, as the goal of classifying hateful speech is very specific. In addition, the self-supervised training of such a model would run into the previously mentioned problems of required time, computational power, and CO2 emission. All these reasons make it unrealistic for an unsupervised NLP model to contribute to this scenario.

**1.2 You define data mining success criteria**

As mentioned before, the business goal is a large, overarching goal that is unlikely to be completed in a single sitting. In addition, its language is not as concrete, allowing for multiple interpretations and possible solutions. Due to all the performed research, however, it’s now possible to create concrete and clear data mining goals to work on during the project. It’s clear where data will be collected, which model will be utilized, and the forms that data will need to take. These data mining goals should also have a focus on the deliverance of clear and tangible products. The currently defined data mining goals that should satisfy the business goal and criteria are:

* Implement a supervised NLP model to classify different speech types
* Create an environment wherein the NLP model’s results can be displayed and reviewed

To ensure that the data mining goals are accomplished successfully, several success criteria can be created in a similar fashion to the business goal’s criteria. Currently, these success criteria are as follows:

* The NLP model has been fine-tuned on additional data
* The NLP model can classify between normal, offensive, and hateful speech
* The NLP model has a 90% + accuracy with its classifications
* The custom /pol/ dataset has been classified using the NLP model
* The results of the classification can be viewed in a Power BI dashboard

It has been mentioned that the Hugging Face models are usable right after importing as they have been previously trained. However, it’s possible that they may not be achieving adequate results right away. This is where the fine-tuning process comes into play, which will make the checkpoint more accurate in theory. This is a complicated process that will be expanded upon more at a later point in this document. Fine-tuning is heavily recommended, making it an interesting task to perform. The result of this criterion is logically a fine-tuning model that can make more accurate classifications.

The chosen Hate-speech-CNERG/bert-base-uncased-hatexplain checkpoint, as well as the additional data that it will be fine-tuned on, have 3 total probabilities to work with. These are normal speech, offensive speech, and hate speech. Certain types of speech are much more inflammatory than others, making it useful to make distinctions between mean comments and deep-seated hatred. The result of this criterion is a model that is much more accurately focused on separating hate speech from other types of speech, instead of conflating everything with a negative context together.

It’s also important for the model to achieve a high accuracy percentage. If the model is unsure about its decisions, then those decisions are unlikely to be trustworthy. There will always be types of speech and sentence structures that are more difficult to classify, but these should be outliers and not the norm. It is unknown beforehand what kind of percentage will be achieved, but accuracies lower than 80% to 90% are less than desired. The expectation of this criterion is a model that can accurately classify, making it trustworthy to use for classifying custom /pol/ dataset as well.

The /pol/ dataset will need to be classified as well. It’s unknown how much data can be collected at this point, but a high quantity is desired. The utilization of NLP models tends to be heavy on computational resources, making this a point to consider. The result of this criterion is an important one, as it relates to the wish of understanding a hateful community. After its realization, it’s possible to create visualizations of the community’s prevalence of hate.

The final criterion is about a Power BI dashboard. The results of this project need to be visualized in some environment to show to the stakeholders. Power BI has been chosen as it is a familiar tool and research has been performed into its visualization capabilities. This program will allow for the creation of complex visualizations such as choropleth maps with much ease. The result of this criterion will be a clear and concise dashboard wherein all the various findings of the project can be found.

One important caveat present in these success criteria is the goal of reaching a 90% classification accuracy, which is perhaps the most important success criteria of all. The HateXplain checkpoint is pre-trained on labelled data and will require input for fine-tuning to be identically labelled as well. These labels originate from human-defined standards about what constitutes offensive language or hate speech. In addition, an external service such as Amazon Mechanical Turk has been used to label the data. Within the confines of this project, there are no similar luxuries. Manually labelling the /pol/ dataset would be impractical both in terms of work volume, and in terms of not being able to align with the correct philosophy from which those labels originate in the first place. For this reason, it’s important to reach an as high as possible percentage of accuracy during the fine-tuning phase for the checkpoint. This will allow the checkpoint to be deemed as trustworthy, even when released on completely fresh data such as the /pol/ dataset.

In terms of metrics for classification problems, there are various metrics that can be utilized. Hugging Face has many metrics available that can be utilized by importing them through datasets library. The chosen metrics to look at for this project are:

* Accuracy
* F1
* ROC AUC

Accuracy is the proportion of correct predictions among the total number of cases processed. It can be computed with: Accuracy = (TP + TN) / (TP + TN + FP + FN) where TP is a true positive, a TN is a true negative, a FP is a false positive, and a FN is a false negative. This metric is not multi-variable and will use the chosen labels the checkpoint is most confident in. Using this metric will give an early look at how well the checkpoint can predict the correct labels. This metric is possible to use with a number of labels higher than 2.

The F1 score is the harmonic mean of precision and recall. According to the metric, this can be computed with F1 = 2 \* (precision \* recall) / (precision + recall)**.** Precision is defined as TP / (TP + FP), where TP is a true positive, and FP is a false positive. Recall is defined as TP / (TP + FN). Precision refers to how many of the positive outcomes were true positives and not false positives. Recall (or sensitivity) refers to how many of the positive outcomes were true positives, and not the result of negative outcomes that were falsely deemed to be positive. This metric is possible to use for both binary and multi-class inputs by using the Hugging Face implementation. Just like accuracy, this metric gives a measure of accuracy, but calculated in a different way.

The ROC AUC metric calculates the Area Under Curve (AUC) for the Receiver Operating Characteristic Curve (ROC). The return values represent how well the model used is predicting the correct classes, based on the input data. A score of 0.5 means that the model is predicting exactly at chance, e.g., the model’s predictions are correct at the same rate as if the predictions were being decided by the flip of a fair coin or the roll of a fair die. A score above 0.5 indicates that the model is doing better than chance, while a score below 0.5 indicates that the model is doing worse than chance. Thus, the ROC AUC metric is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. This metric is also possible to use on multi-class problems.

In the end, all these metrics will return a measure of how well the checkpoint is performing. However, they all use different methods of calculating their respective outcomes, making it possible that the checkpoint could be performing well on one front, but do poorly on another. Each of the metrics thus focus on different characteristics of the checkpoint, making it useful to perform multiple of such metrics at the same time.

**1.3 You add extra self-organized and/or external data sources to the Data Science process**

Within the context of this project, there is no initial data granted by the stakeholders due to how this project is very exploratory in its nature. To fully realize this project, two datasets need to be collected. The first is the Davidson et al. (2017) dataset, originating from the HateXplain research paper. The second is data from the /pol/ community through the utilization of one of the 4chan APIs.

The first set of data is easily collected, as it can be found on the hate-speech-and-offensive-language GitHub repository of t-davidson (Davidson, 2017). By downloading the labeled\_data.csv file, this data has already been acquired. A preview of this data is shown in the following figure 11. Do note that it might contain strong and offensive language.

Diagram

Description automatically generated with low confidence

*Figure 11: Preview of the t-davidson dataset*

This data contains various columns, whose meanings are explained in table 3.

|  |  |
| --- | --- |
| Feature | Description |
| Count | Number of CrowdFlower users who coded each tweet (min is 3, sometimes more users coded a tweet when judgments were determined to be unreliable by CF |
| Hate\_speech | Number of CF users who judged the tweet to be hate speech. |
| Offensive\_language | Number of CF users who judged the tweet to be offensive. |
| Neither | Number of CF users who judged the tweet to be neither offensive nor non-offensive. |
| Class | Class label for majority of CF users. 0 - hate speech 1 - offensive language 2 - neither |
| Tweet | The text that was judged by the CrowdFlower users. |

*Table 3: Features and descriptions of the t-davidson dataset*

From looking at the data, it’s clear that the 3 desired labels are indeed present in the dataset. The dataset is also about tweets, which makes it unlikely for certain messages to contain long and unintelligible rants due to Twitter’s inherent character limit. CrowdFlower is a service that can be used for the classification of texts, tweets in this case. This service is like Amazon Mechanical Turk, which has been used for the HateXplain model. This data is vital for the Data Science process due to it being labelled identically to the original HateXplain data that was used for its pre-training process. To properly fine-tune a checkpoint, it’s necessary that any further given data is given in the same format, and with the same contents. If this is not the case, completely unexpected results may occur that are impossible to trace. Without this fine-tuning of the checkpoint to reach a high percentage of accuracy, it will be unknown how well the checkpoint is potentially capable of performing. Without this guarantee, it will be unknown whether the custom /pol/ dataset can be properly classified as well. This dataset also has a direct connection to the success criteria of attaining a classification accuracy of at least 90%, making it a critical piece for the successful completion of this Data Science process.

Collecting data for the /pol/ dataset is more of a challenge due to the requirements of using an API. The API suitable for this task is the 4chan-API, whose documentation is found on in a GitHub repository as well (4chan, n.d.). This section will contain many terms and abbreviations which are likely to be difficult to understand without additional explanation. Before handling the API, some more context needs to be given about 4chan and its /pol/ community. Note that any unfamiliar terms can be found in the Terminology section of this document.

As the project concerns itself with only a particular section of the website, it’s not necessary to expand upon the core of the website itself. Since posting is anonymous, no signing up for an account is necessary, and all sections of the website can be visited directly without further issue. Note that while this section will contain figures that depict the information being given, any strong language or NSFW (Not Safe For Work) imagery will be censored. Though expletives may be censored, any topics visible may still contain an offensive nature. For this reason, a direct link to the website will not be supplied either. Figure 12 shows what one typically sees after venturing into the politically incorrect board for the first time.

Graphical user interface, text

Description automatically generated

*Figure 12: Landing page of 4chan’s /pol/ board*

Visible within the figure are three threads, of which the first two are referred to as a “sticky”. This means that the thread has been pinned at the top of the board by the website’s moderators, making them immune to expiration or deletion. This is used for the creation of announcements or otherwise large events that would otherwise flood the entire board with similar threads. Aside from the stickies, any other thread can be visited and replied to. It’s also possible to press the catalogue button to have an overview of all the currently active threads, as visible within figure 13.

Timeline

Description automatically generated

*Figure 13: Catalogue of threads on 4chan’s /pol/ board*

As shown, there are many threads active at once. By visiting a thread and examining a typical response to said thread, a clearer picture can be given about what kind of data can be collected. Figure 14 holds a response to a thread with many of these typical elements.

Graphical user interface, text

Description automatically generated

*Figure 14: The standard makeup of a response in a thread on 4chan*

Visible within figure 14 is a large portion of the data available when collecting data from this website. It’s shown that the poster is labelled as Anonymous, which is the standard when posting due to the lack of account creation. It is possible to fill out a name when posting, but it’s not unique, and other posters are able to supply the same name as well. The exception to this rule is a tripcode, which is a type of pseudo-registration that can help verify a user. This is done by supplying a combination of a name and another word or phrase, separated by a hashmark (“#”). Upon submission, the server will generate the hash unique to that specific word or phrase. For example, a combination of User#password would turn into User !ozOtJW9BFA. However, using a tripcode is generally frowned upon by other users and they are seldomly used. Next to the username are two features that are not used in every board: the ID and the country flag. The ID is an alphanumeric string and a background colour a user receives when posting a thread, or in a thread. When said user posts in that thread once again, the ID will be the same. The ID is tied to the IP address of the user, and it does not carry over between different threads. The country flag depends on geographical location and allows users to see which country everyone comes from. In addition, it’s possible to manually choose several non-country flags as well, such as flags representing beliefs such as anarchism, communism, or even Nazism. However, similarly to the tripcode, usage of these flags is frowned upon as well and referred to as using a “memeflag” under the belief that the user is afraid of being ridiculed for their country of origin. Next to the country flag are the date and time when the post was made, and to the right of that is the post number of the reply. Every single time a post is made, the counter of this number goes up by one, and it serves as the ID of replies for database purposes. As each number can only be achieved once, certain combinations, such as posts ending in repeated digits, or every millionth post, are seen as very special. These posts are usually referred with terms like “gets”, or “dubs”, “trips”, “quads”, etc., for posts ending in two, three, four, or more repeated digits. Visible within the image is that the number of replies on this board is already up in the 380 million, solidifying its position as one of the most popular boards on 4chan. The user in this reply has decided to attach an image, of which the filename, file size, and dimensions are shown.

The actual content of the post contains various elements that are commonly seen as well. Firstly, the user refers to another reply in the thread by typing out its post number and prefixing it with two right-facing arrows (“>>”). This allows users to directly reply to certain posts to prevent confusion and clicking the numbers will show the post that has been replied to. In this case, the user in the image has replied to the user that originally created the thread. For this reason, the direct response shows (OP) next to it. OP stands for Original Poster, and it always refers to the creator of the thread. Secondly, two more colours of text are visible within this user’s reply. Red is the standard colour for regular text, but green-coloured text is created by prefixing the line with a single right-facing arrow symbol (“>”). Aside from the colour change, it has no direct functionality, but it is used as a manner of quoting things other users have said, or for telling stories that have happened. Storytelling using this feature is often referred to as “greentexting” or writing a “greentext”.

It's clear that a single reply already holds a lot of information that can be processed by all kinds of data science models. At first glance, however, the collection of data may seem like an arduous task due to the fluid state of threads, and how they expire automatically. However, archiving services exist and 4chan also has its own API. The API can be utilized to download thousands of threads in .json format. Visible in figure 15 is an example of the .json data of a thread on the origami board /po/ (not to be confused with /pol/).

Text

Description automatically generated

*Figure 15: .json data showing the first reply of a thread on 4chan’s /po/*

Many of the elements that were just identified in figure 14 are also immediately apparent in this figure. Examples of this are the data and time, the username, the comment itself, attached file information, etc. However, many more data fields exist depending on various circumstances, which makes the .json info size of each reply not uniform. All the possible data fields are expanded upon in the following table 2. The documentation in the GitHub repository also supplies more information such as the data type, when it appears, and example values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attribute | Type | Appears | Description | Possible Values |
| no | integer | always | The numeric post ID | any positive integer |
| resto | integer | always | For replies: this is the ID of the thread being replied to. For OP: this value is zero | 0 or any positive integer |
| sticky | integer | OP only, if thread is currently stickied | If the thread is being pinned to the top of the page | 1 or not set |
| closed | integer | OP only, if thread is currently closed | If the thread is closed to replies | 1 or not set |
| now | string | always | MM/DD/YY(Day)HH:MM (:SS on some boards), EST/EDT timezone | string |
| time | integer | always | UNIX timestamp the post was created | UNIX timestamp |
| name | string | always | Name user posted with. Defaults to Anonymous | any string |
| trip | string | if post has tripcode | The user's tripcode, in format: !tripcode or !!securetripcode | any string |
| id | string | if post has ID | The poster's ID | any 8 characters |
| capcode | string | if post has capcode | The capcode identifier for a post | Not set, mod, admin, admin\_highlight, manager, developer, founder |
| country | string | if country flags are enabled | Poster's ISO 3166-1 alpha-2 country code | 2-character string or XX if unknown |
| country\_name | string | if country flags are enabled | Poster's country name | Name of any country |
| board\_flag | string | if board flags are enabled | Poster's board flag code |  |
| flag\_name | string | if board flags are enabled | Poster's board flag name | Name of a board flag |
| sub | string | OP only, if subject was included | OP Subject text | any string |
| com | string | if comment was included | Comment (HTML escaped) | any HTML escaped string |
| tim | integer | always if post has attachment | Unix timestamp + microtime that an image was uploaded | integer |
| filename | string | always if post has attachment | Filename as it appeared on the poster's device | any string |
| ext | string | always if post has attachment | Filetype | .jpg, .png, .gif, .pdf, .swf, .webm |
| fsize | integer | always if post has attachment | Size of uploaded file in bytes | any positive integer |
| md5 | string | always if post has attachment | 24-character, packed base64 MD5 hash of file |  |
| w | integer | always if post has attachment | Image width dimension | any positive integer |
| h | integer | always if post has attachment | Image height dimension | any positive integer |
| tn\_w | integer | always if post has attachment | Thumbnail image width dimension | any positive integer |
| tn\_h | integer | always if post has attachment | Thumbnail image height dimension | any positive integer |
| filedeleted | integer | if post had attachment and attachment is deleted | If the file was deleted from the post | 1 or not set |
| spoiler | integer | if post has attachment and attachment is spoilered | If the image was spoilered or not | 1 or not set |
| custom\_spoiler | integer | if post has attachment and attachment is spoilered | The custom spoiler ID for a spoilered image | 1-10 or not set |
| replies | integer | OP only | Total number of replies to a thread | 0 or any positive integer |
| images | integer | OP only | Total number of image replies to a thread | 0 or any positive integer |
| bumplimit | integer | OP only, only if bump limit has been reached | If a thread has reached bumplimit, it will no longer bump | 1 or not set |
| imagelimit | integer | OP only, only if image limit has been reached | If an image has reached image limit, no more image replies can be made | 1 or not set |
| tag | string | OP only, /f/ only | The category of .swf upload | Game, Loop, etc.. |
| semantic\_url | string | OP only | SEO URL slug for thread | string |
| since4pass | integer | if poster put 'since4pass' in the options field | Year 4chan pass bought | any 4-digit year |
| unique\_ips | integer | OP only, only if thread has NOT been archived | Number of unique posters in a thread | any positive integer |
| m\_img | integer | any post that has a mobile-optimized image | Mobile optimized image exists for post | 1 or not set |
| archived | integer | OP only, if thread has been archived | Thread has reached the board's archive | 1 or not set |
| archived\_on | integer | OP only, if thread has been archived | UNIX timestamp the post was archived | UNIX timestamp |

*Table 4: All possible .json attributes available through the 4chan API*

It’s clear that an abundance of data is available in even just a single post. However, many of the features are unlikely to be of any relevance to this project’s goals, such as the features relating to image dimensions. Many of the features are not always available as they are contingent on factors such as the reply being the OP, whether a thread is archived, etc. Features that are likely to be relevant to this project’s data mining goals and criteria are “no”, “resto”, “now”, “time”, “country”, “country\_name”, “com”, and “replies”. The other values can likely be discarded. The collection of this data is handled in an upcoming section of this document. In terms of the actual modelling process, only the “com” feature will be necessary due to the way a sequence classification NLP model works. However, the other chosen features remain useful for an initial exploration of the data’s contents.

This data is of critical value to the data mining goals as well as it is the data relating to the environment that The Equal Web wishes to research. If this data were to be unavailable, then the related community Is unavailable, or much harder to research. In that scenario, it would be necessary to go back to the drawing table to seek another approach, or another community. Naturally, this data has a direct connection with the success criteria of classifying the custom /pol/ dataset. Without any data, this objective would be impossible to complete.

In the future, it’s possible to expand upon this project in various ways. The first option is to increase the amount of data to look at. With the current /pol/ dataset, while it contains a large quantity of sequences already, only 4 days’ worth of data is present. It’s likely that the usage of hateful speech is heavily tied to certain topics. It’s possible for certain periods of time to draw more attention to topics that are more inflammatory than others. Seeing how most of the user traffic is likely to come from America, one such example would be the American presidential elections. For this reason, it could be useful to collect data that spans a larger period. This can once again be done manually, or it’s possible to seek out existing archives of 4chan-related data. One such source is Papasavva et al. (2020), which contains 134.5 million user posts of the /pol/ board, spanning a period of 3.5 years in total (June 2016-November 2019). Analysing the data may yield interesting patterns of upticks or downturns in hateful speech. It would also be possible to expand the scope to other boards as /pol/ is far from the only one where these hateful phenomena take place.

Another option would be related to a different NLP task of optical character recognition to detect text in images. Very often, replies are submitted that contain no text but do contain an image. Within the context of this project, these have been removed as the project only focused on text. However, many of these images are often infographics and rich in information related to inflammatory topics or conspiracy theories, or they contain news headlines and articles in line with the board’s darker sense of humour. Such images can be infinitely richer in subject matter than a character-limited reply could possibly hope to be. In this case, it would be possible to use the same API method to collect data, but for the purpose of collecting the attached images instead. Collecting a large quantity of suitable images would be able to provide additional information in common thought processes and rationales that circulate around the board.

Learning Objective 2: You collect and address relevant data

**2.1 You (re-)validate data after model-generated assumptions**

Data first needs to be collected from the 4chan API, as mentioned. This can be performed by using the code shown in figure 16.



*Figure 16: 4chan API connection and data collection*

A connection is made to the 4chan API by going to the archive of the API and receiving the json. The archive is where threads that have recently been deleted are temporarily stored. The archive on the website itself states that the most recent 3,000 threads are available, but more than 12,000 threads worth of replies are being collected, as visible in the image. As not every item in the API will contain the same number of columns (due to some only appearing under certain circumstances), it’s necessary to work with DataFrames to ensure that data is always saved in the correct location. For this reason, the order DataFrame is first made, containing columns with names mirroring all the fields in the 4chan API. The json for the archive has only rudimentary information about the saved threads, making it necessary to visit each thread individually within the API to retrieve all the necessary information. This is happening within the loop, where the data of the thread is retrieved, saved in a temporary DataFrame, concatenated to the order DataFrame, and then saved to an external CSV file. The order DataFrame is emptied afterwards, and the final print statement is for keeping track of progress of the process. The code in this figure fetches all the features as it was unknown at that point which features exactly will be required for the entire Data Science process.

By using the Pandas library, the CSV file can be loaded into Jupyter Notebook, and by specifying the exact desired columns as well. As mentioned before, the chosen features are “no”, “resto”, “now”, “time”, “country”, “country\_name”, “com”, and “replies”. The results are visible within figure 17.

Text

Description automatically generated

*Figure 17: Loading and head() function of the /pol/ dataset*

The head() function already reveals several things about the dataset. First, the “resto” feature is 0 when the specified post is the OP of a thread, otherwise it refers to the post number of that OP instead. The “now” feature contains information about the date when a reply was posted, but it contains superfluous information about the exact time. The “time” feature is a UNIX timestamp which can be decoded into a regular timestamp. A decision will need to be made about which feature to use. The “com” feature, which contains the user messages, contains NaN values and all kinds of HTML codes that will need to be cleaned. The “replies” feature only contains a value when a post is the OP, designating the number of replies in that thread. This is unlikely to be as useful as expected and could possibly be discarded. Overall, all the feature names could use a renaming, as they are not very descriptive as of now. An info() function can be called to reveal information about data types, dataset size, and number of missing values. This is shown in figure 18.

Graphical user interface, text

Description automatically generated

*Figure 18: info() function of the /pol/ dataset*

Based on this figure alone, it’s possible to acquire all kinds of information, such as the generic data types and sub-types of the features. Based on the Dtypes that the info() function provided, these are shown in table 5.

|  |  |  |
| --- | --- | --- |
| Feature | Generic Data Type | Sub-type |
| No | Quantitative | Discrete |
| Resto | Quantitative | Discrete |
| Now | Qualitative | Ordinal |
| Time | Quantitative | Discrete |
| Country | Qualitative | Nominal |
| Country\_name | Qualitative | Nominal |
| Com | Qualitative | Nominal |
| Replies | Quantitative | Continuous |

*Table 5: Generic data types and sub-types of the /pol/ dataset features*

The “no”, “resto”, “time”, and “replies” features are of the quantitative data type due to consisting of numerical values. The “replies” feature is of the continuous type as this column holds the number of replies each individual thread had, which can be any number and is not influenced by anything else. The other features are of the discrete sub-type as they are confined by strict logic. The post numbers serve as unique identifiers and thus adhere to a system where each new reply increments the number by 1. There are no gaps in this system. The “time” feature is discrete as well as its numerical contents are conversions of UNIX timestamps. As they can be converted back as well, they are confined by a certain logic too.

The “now”, “country”, “country\_name”, and “com” features are of the qualitative variety due to consisting of categorical data. Aside from “now”, they are all of the nominal sub-type. The “now” feature is ordinal as it contains dates and times. As the post number increments, so will the time at which that reply is posted. It’s impossible for the timestamp of a post to come before the timestamp of another while the post number is higher. The other features are nominal as they contain categorical values where no order persists.

It’s also visible that data is missing from some of the features as their non-null count is not equal to the 494,345 total entries in the dataset. These numbers of missing values can also be further inspected with the code from figure 19.

Text, table

Description automatically generated

*Figure 19: Missing values of the /pol/ dataset*

It’s quite easy to figure out the reasons behind why there is data missing in these features. The missing data reasons and explanations are shown in the following table 6.

|  |  |  |
| --- | --- | --- |
| Feature | Missing Data Reason | Explanation |
| No | None! | No data is missing |
| Resto | None! | No data is missing |
| Now | None! | No data is missing |
| Time | None! | No data is missing |
| Country | Missing at random (MAR) | Using an alternative flag obfuscates the geographical location flag |
| Country\_name | Missing at random (MAR) | Using an alternative flag obfuscates the geographical location flag |
| Com | Missing at random (MAR) | Replies may be devoid of text and only contain an image |
| Replies | Missing not at random (MNAR) | This value is only present for the reply that counts as an OP |

*Table 6: Missing data reasons and explanations of the /pol/ dataset*

For the first 4 features, there is no data missing at all. For the others, data is missing but the reasons behind them are innocuous. Using an alternative flag on the board overwrites the flag that would have shown regularly to represent the country where a user is geographically posting from. While the cause is known, this happens randomly as this is something that people sometimes decide to do. There are no specific individuals, rules, or patterns to attach to this reason. The same holds true for the “com” feature. People may simply choose to only reply with an image. There are no patterns to be found here as well. The “replies” feature is only filled in when the related post counts as an OP: the one that started a thread on the board. It shows the exact number of replies that thread has gotten. This missing data thus originates from an automated process, making the mechanism behind the missing data known. How to handle these missing values is handled at a later point in the document.

As stated, there are 494,345 total entries, which will surely be enough data to use for the model. Even after a reduction through the removal of missing values, plenty of data will remain. A simple nunique() can be called on the “no” feature of the dataset to see how many messages are available, as shown in figure 20.

Graphical user interface, text, application

Description automatically generated

*Figure 20: Unique values inside of the “no” feature of the /pol/ dataset*

It’s also possible to look at the period wherein this data was harvested, which is done through the code in figure 21.

Text

Description automatically generated

*Figure 21: Period of time of the /pol/ dataset*

The figure shows that the data collected from the /pol/ archives spans about 4 days. As the project does not concern itself with the passing of time, this is no issue. However, a broader period might be preferred during a different project to consider periodic events that could potentially spark higher or lower influx of messages and hateful content. In addition, it can be surmised from the previous info() function that there are 11,812 unique threads present in this dataset. This “replies” feature can be further inspected as visible in figure 22 and visualized in a boxplot according to figure 23.

Text, table

Description automatically generated

*Figure 22: Statistical summary of the /pol/ dataset’s “replies” feature*

A picture containing box and whisker chart

Description automatically generated

*Figure 23: Boxplot of the /pol/ dataset’s “replies” feature*

All this information shows that many of the threads that pass the board do average in replies. Some of the topics seem to be uninteresting and are quickly replaced by other threads as they are eventually deleted from inactivity. Other threads are incredibly popular, breaking even the 400 or 500 reply count. These findings are of no consequence to the actual modelling that will be performed, but it could be a starting point for future research to investigate which topics are the most popular on a board.

It’s also interesting to look at the countries and see which are the most prevalent in terms of board activity. According to the following figure 24, there are 157 unique countries present in the dataset.

Graphical user interface

Description automatically generated with medium confidence

*Figure 24: Number of unique countries in the /pol/ dataset*

Plotting this number of countries would make any plot unintelligible, making it more interesting to look at the top X countries instead. In addition, it’s likely that many of the bottom countries are the result of users messing around with VPNs to spoof their location. A pie chart of the top 10 countries by traffic can be created as according to the code in figure 25.

Chart, pie chart

Description automatically generated

*Figure 25: Top 10 countries by total board traffic*

To no surprise, the United States dominates most of the board in terms of total traffic. After that, the ranking is mostly comprised of European countries, with the Netherlands also making the top 10.

The total traffic of this top 10 can also be compared against the total number of rows available in the dataset, which is done as shown in figure 26.

Graphical user interface, text, application

Description automatically generated

*Figure 26: Percentage of total rows in the dataset comprised of the top 10 countries by volume*

A total of about 73% of all posts within this 4-day period came from these top 10 countries. As the period encompasses multiple days, every side of the world has had their chances to contribute, making it an even contest. Naturally, total traffic correlates to population of each country. In terms of further analysis and statistical summaries, the /pol/ data is not that exciting, and it has little to offer that has any influence on the tasks at hand. All the main issues that will need tending to relate to missing data, the renaming of columns, and the improvement of the textual quality of the user replies to cater to the model’s expectations.

However, the model will have to undergo a fine-tuning process for which the t-davidson dataset is supplied. This makes it necessary to inspect this dataset as well, which is first imported according to figure 27.

A picture containing graphical user interface

Description automatically generated

*Figure 27: Reading of the t-davidson dataset*

The usage of the index\_col argument is necessary to avoid an unnecessary Unnamed: 0 index column. Immediately afterwards, a head() function can once again be called to peek at the dataset’s contents. The results are visible in figure 28.

Table

Description automatically generated

*Figure 28: Results of the head() function of the t-davidson dataset*

It can be noted that the first 4 columns can be immediately disregarded for this project’s purpose. These columns related to the number of CrowdFlower participants that have worked on labelling each sequence, and how many votes for each category that sequence received. The “class” feature is the actual label that the sequence received, and the “tweet” feature is the comment that was judged by the CrowdFlower participants.

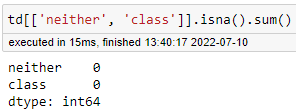
Once again, to ensure that the data does not contain any quirks that need alteration, various aspects of the selected features can be examined. Table 7 contains the data’s generic data types and sub-types.

|  |  |  |
| --- | --- | --- |
| Feature | Generic Data Type | Sub-type |
| Class | Quantitative | Discrete |
| Tweet | Qualitative | Nominal |

*Table 7: Generic data types and sub-types of the t-davidson dataset features*

The “class” feature contains only the values 0, 1, and 2, as mentioned. This makes the feature quantitative and discrete due to originating from the logic of CrowdFlower judgement. As the “tweet” feature contains textual comments, it’s of the qualitative type. It also has the nominal subtype due to its contents being completely random.

To fully ensure that there is no data missing either, the code from figure 29 can be executed.



*Figure 29: Missing values of the t-davidson dataset*

Thus, the missing data reasons are as shown in table 8.

|  |  |  |
| --- | --- | --- |
| Feature | Missing Data Reason | Explanation |
| Class | None! | No data is missing |
| Tweet | None! | No data is missing |

*Table 8: Missing data reasons and explanations of the /pol/ dataset*

As no data is missing, there is nothing more to say about this topic. The dataset is in good condition so far, making it suitable for the purpose of fine-tuning the HateXplain checkpoint. According to the GitHub wherein the t-davidson dataset was located, the label 0 correlates to hate speech, 1 to offensive language, and 2 to neither. While the label categories are the same as those in the HateXplain model (albeit slightly differently named), the offensive language and normal speech labels are switched around, making this swap a necessary preparation step in the future. The code in figure 30 can be used to look at the distribution of speech types within the “class” feature.

Graphical user interface, text

Description automatically generated

*Figure 30: Distribution of speech types within the t-davidson dataset*

This means that 19,190 of the 24,783 total sequences contain offensive language, 1,430 contain hate speech, and 4,163 of them contain what was deemed to be speech belonging to neither category. It’s also visible in figure 30 that many of the tweets contain HTML elements and codes that are also present in the /pol/ dataset. These will need to be escaped and/or removed as well. The distribution of label types seems uneven at first, but this might not be an issue for the HateXplain checkpoint. The reason behind this is that the checkpoint has already been pre-trained, it knows what offensive speech and hate speech are like for the most part. The number of hate speech present within the t-davidson dataset is not negligible and will help to improve the checkpoint’s accuracy.

It’s now possible to talk about 3 important topics to judge whether the available data will be sufficient for the upcoming modelling process. Note that the /pol/ dataset will be used after the modelling process, and the t-davidson dataset during the process. The 3 topics are:

* Representativity
* Independence
* Bias

It’s important for data to be representative. What this refers to is that samples of the data accurately reflect the characteristics of the whole. Within the context of a sequence classification NLP project, the requirements are quite simple. The data contains 3 labels, making it necessary that any sample also contains an ample amount of each label. For the fine-tuning of the checkpoint, it will be necessary to split the data between a training, a validation, and a testing set of data. Usually, this splitting is done randomly to take samples from all over the dataset. However, when one of the labels is underrepresented in comparison to some of the other labels, it could occur through a stroke of bad luck, that one of the splits takes a low number of one of the labels. Without noticing this occurrence, it could severely impact the fine-tuning of the checkpoint. This can be remedied by performing the splitting of the data in a stratified fashion. Stratification implies the splitting of the data in a fashion that ensures equal representation of all the labels. Making use of this option would clear up any worries of underrepresentation or overrepresentation.

The topic of independence relates to how data isn’t dependent on other types of data that are not accounted for in the chosen model. For example, how much money you earn is dependent on how much you work. Other factors are at play as well, but it’s certain that there are dependencies present. There are two types of independence to consider, which are:

* Independence for observations within a group
* Independence within each group

Independence for observations within a group relates to how observations between groups should be independent. Within the context of the t-davidson dataset, it means that each split after the splitting of the dataset contains fully unique observations. One or more of the comments appearing multiple times between different sets of data would be able of skewing the results in a certain direction. It’s important to avoid this kind of dependence as much as possible to ensure that each set of data remains devoid of such impurities. Independence within each group refers to how data points within each group should be unconnected as it could otherwise skew the data. One data point could be the cause of another data point existing. This simultaneously means that they skew the data by coexisting, and that severing that connection would throw off the data as well. An example would be the sales numbers of ice cream, which has a seasonal component to it. More ice cream is sold in the summer than in the winter. These dependencies need to be accounted for in the model of choice. Within the context of the modelling for this project, only the user messages and labels are relevant features that will be given to the checkpoint. Other than the innate connection between certain types of speech and the given labels, there are no other dependencies to consider when designing the checkpoint.

The topic of bias relates to how an algorithm can produce results that are systematically prejudiced due to incorrect assumptions in the machine learning process. The most egregious example of this relates to topics such as race, gender, religion, etc. To use a contemporary example, many police forces around the world are experimenting with predictive policing: using AI to predict when or where crim will happen and predicting whether high-risk groups or individuals may commit a crime in the future. Without proper care, predictive models such as these may exhibit behaviour and results that are prejudiced against people of certain descent, belief, faith, etc. It’s always necessary to consider these biases that might occur, both in AI and in your own though processes. There are many biases, but they are usually unintentional and accidentally introduced into a machine learning model. Within the context of this project, there are no further identifying factors available that may introduce bias into the model as no groups or individuals can be tied to any of the comments. However, it’s important to note that specific types of speech, and who can use these types of speech, are highly contested topics nowadays. An example is African American Vernacular English (AAVE), which may be deemed offensive by some if used by a non-African American individual, and not by others. In addition, context always matters, and these single sequences have no further context to offer. Due to the lack of context and who has said what, the sequences can only be judged based on their own merit. This has been done by the CrowdFlower participants, based on a list of criteria that was given to them. For this reason, all the sequences have been judged in the same manner, making them uniform in rationale and free of other limiting factors.

Thus, the dataset is representative, independent, and free of bias, making it a good candidate for furthering the improvement of the HateXplain checkpoint.

**2.2 You integrate relevant data by merging multiple data sources**

Chronologically, the submitted proof for this section of the document takes place after the modelling operations performed in section 3.3. This section may thus contain code and concepts that are explained at a later point, and it might be better to return to this section later to fully understand what is going on. At this point, the /pol/ dataset has been cleaned and the fine-tuned checkpoint can be used to classify sequences that are available in the dataset. Due to constraints related to memory and computational power, the resulting labels are stored in an external CSV file. After everything is done, this file needs to be integrated into the project and merged with the original /pol/ dataset.

The cleaned /pol/ dataset has been saved before, and it can now be imported by using the Pandas library’s read\_csv() function, as shown in figure 31.

Graphical user interface, text, application

Description automatically generated

*Figure 31: Loading of the cleaned /pol/ dataset*

In addition, it’s shown that the dataset contains 455503 total rows and 7 columns. The head() function also gives a preview of the current state of the data, indicating that the data is in good form and that the cleaning process of the comments has been performed successfully. Like with the modelling process, the checkpoints need to be loaded first, which is done with the code in figure 32.

Text

Description automatically generated

*Figure 32: Loading of the sequence classification checkpoints*

The original checkpoint is loaded as well to grab the tokenizer from, and then the fine-tuned checkpoint is loaded from where it was saved on my PC.

The usage of the fine-tuned checkpoint takes a large toll on the memory usage of my machine, with the classification of about 500 sequences already maxing out 32GB of memory and causing the machine to freeze. For this reason, a loop that works around this constraint has been constructed, whose implementation can be seen in the following figure 33.

Text

Description automatically generated

*Figure 33: Loop to label all /pol/ dataset sequences*

The loop will work in batches of 100 sequences. For this reason, the total length of the /pol/ dataset is divided by 100 and rounded up, resulting in a value of 4556. This is how many times the loop will have to go through the process of classifying 100 labels. The start and batch variables within the loop ensure that the proper indices are selected for each new batch. This way, no duplicates will appear in the resulting data as well. The code then goes through the process of tokenizing, classifying with the model, converting the output to legible percentages, and saving the results in a DataFrame called modelresults. This DataFrame already contains 3 rows, each containing one of the 3 labels and a confidence value on each of its rows. A new column called “label” is created to store the chosen label by the checkpoint, which is the label with the highest percentage value. This is done by making use of the idxmax() function and then converting the chosen label names to their respective numeric values. A second column called “post\_number” is created as well, containing the post numbers of the current batch’s sequences. This column is important for merging the datasets, and to determine if any data will be missing.

The results are then stored within the polplabels.csv by appending them. This way, no data gets unintentionally overwritten. Afterwards, the batch, token, output, predictions, and modelresults variables are deleted to free up memory. As they are recreated during every iteration of the loop, this will prevent memory issues and ensure that nothing goes wrong during the process. By keeping track of the memory usage within the task manager, it’s possible to observe the eb and flow of memory usage while the loop is running. This is shown in figure 35.

Chart, line chart

Description automatically generated

Figure 35.

The implementation is working, and the memory is used and freed up every time as expected. Some of the spikes that can be observed are larger than others, which is the case due to the length of input sequences. Longer messages require more tokenization. This is also the reason why any sequence longer than 500 characters has been culled beforehand. By sticking to a batch size of 100, it’s also guaranteed that any batch containing a larger tokenization process will not skirt the 32GB memory limit. While the implementation works, the classification of 455,503 sequences is still a long process and it took about 17 hours to complete.

After completion, with all the data saved in the CSV file, it’s now possible to combine the original /pol/ dataset and the created labels. This done by first importing both datasets into the same environment, using familiar code as shown in figure 36.



*Figure 36: Import of the /pol/ dataset and the /pol/ labels dataset*

Their shapes can be explored to confirm whether the same number of rows is available in both datasets. This is done according to figure 37.

Graphical user interface, text, application

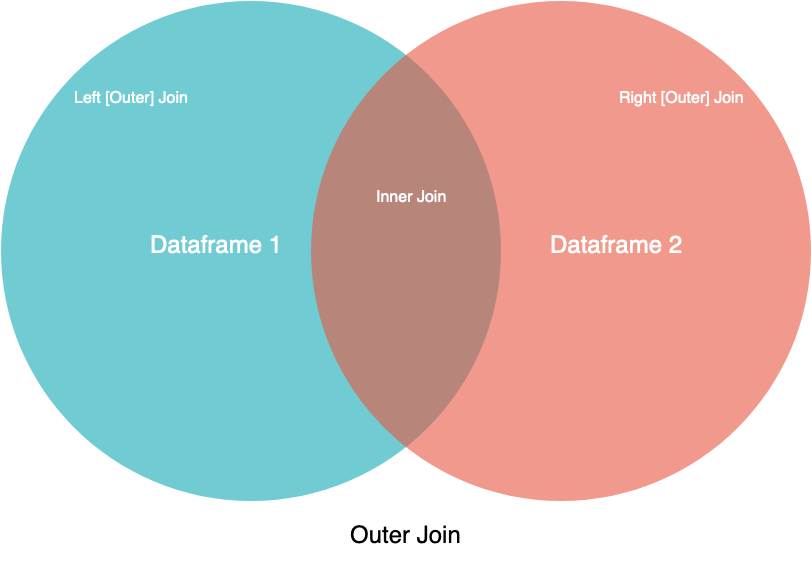
Description automatically generated

*Figure 37: Shapes of the /pol/ dataset and the /pol/ labels dataset*

Combining the datasets together can be achieved in several ways, depending on the desired results. The Pandas library offers three relevant functions here, which are:

* merge() for combining data on common columns or indices
* join() for combining data on a key column or an index
* concat() for combining DataFrames across rows or columns

Merge is flexible and perfectly applicable in this scenario, and it is akin to the functionality of a join operation performed on an SQL database. Usage of this function requires for both a left and right DataFrame to be provided. These terms refer to the two DataFrames that will be combined. The terms left and right refer to which data is kept when combining two DataFrames. The following figure 38 can be used to illustrate this concept more.



*Figure 38: Visual representation of merges/joins*

While simplified, figure YY gives a visual overview of the behaviour with different kind of joins or merges. It’s possible for datasets to be of unequal size. Were they to be combined, how do you deal with issues such as the chosen key column not having 100% equal values, or a different number of values? The left and right arguments determine which dataset is dominant, and whose data takes priority to be kept. Without providing any arguments, merge function will default to an inner join, keeping only rows that have a match in the chosen key columns.

According to https://realpython.com/pandas-merge-join-and-concat/ merge() is a module function and join() is an instance method that lives on your DataFrame. Under the hood, .join() uses merge(), but it provides a more efficient way to join DataFrames than a fully specified merge() call. This enables you to specify only one DataFrame, which will join the DataFrame you call join() on. Concatenation is a bit different from the other two merging techniques. With merging, you can expect the resulting dataset to have rows from the parent datasets mixed in together, often based on some commonality. Depending on the type of merge, you might also lose rows that don’t have matches in the other dataset. With concatenation, your datasets are just stitched together along an axis, either the row axis or column axis.

In this project’s scenario, the data has been prepared in such a way that there is a 100% match between a column that both datasets have in common. For this reason, the merging process is rather simple, and what type of joining function is used does not matter all that much. To illustrate this, two different kinds of merges can be performed. These are performed as shown in figure 39.

Graphical user interface, text, application

Description automatically generated

*Figure 39: Two different merges of the /pol/ dataset and the /pol/ labels dataset*

The merge function is given the two datasets to combine, those being saved in the polp and polplabels variables. For both functions, the “post\_number” feature is selected to be the key column to merge on, as this the column whose name is shared across both datasets, and whose contents are the same as well. For the first merge function, it’s specified to merge as a left join. For the second merge function, no such argument is provided, which will make the function default to an inner join. The results of both merges can be observed in figure 40.

Graphical user interface, text, table

Description automatically generated

*Figure 40: Merge results of the two chosen merge functions*

As visible, both merges have the exact same results. Both results have no missing data, retain the same number of rows, and the same number of columns is present as well. The merging process has been performed successfully. With these results, it’s also possible to throw a quick glimpse at the label results that the fine-tuned checkpoint has come up with. This is done according to figure 40.

Graphical user interface, text

Description automatically generated

*Figure 40: Distribution of types of speech across the /pol/ dataset*

Recall that label 0 refers to hate speech, label 1 refers to normal speech, and label 2 refers to offensive speech. This distribution means that with hate speech and offensive speech combined, about 30% of the 455,503 sequences contains language that is likely to be cause for moderator intervention in any other online environment. In total, these are 134,848 sequences. Naturally, not every topic on the /pol/ board is as inflammatory, and not every thread is as likely to get equal traction. It’s no surprise that harmless comments are in the majority, but the number of hate speech and offensive speech is still staggering.

Now that the datasets have been combined, the whole can be saved in a new CSV file for later use in Power BI. While not all currently present features will be utilized, there is no harm in letting them stay. The saving of this DataFrame is done as shown in figure 41.

A picture containing text

Description automatically generated

*Figure 41: Saving of the polpmerged DataFrame to a CSV file*

**2.3 You clean data by imputing and scaling relevant data**

Imputation is a method of dealing with missing data. It’s a rare occurrence that supplied data is in a pristine state, without the need for any further alterations. A common ailment is the prevalence of missing values within a dataset. It’s possible to simply remove rows with missing values from the dataset, but this may sometimes leave you with a barren dataset unusable for any further operations. Imputation is the process of replacing missing data with substituted values. When done properly, it’s possible to salvage your dataset from the sorry state it was originally in.

Scaling refers to changing the range of your data, which is a process that changes the values of numerical variables so that the transformed data points have specific helpful properties. For example, data may contain scores or ratings that range from 1 to any random integer like 8. Depending on context, it may be unclear what this means at first glance. Scaling could be used to adjust the scale of values this data lies on to create a more legible whole. However, scaling does not change the shape of the data, meaning that the data stays in its original state. There’s also the process of normalization, but this is a more radical transformation that changes the observations so they can be described as a normal distribution. This is usually done when the chosen model is expecting data that is normally distributed.

However, neither imputation nor scaling have had to take place for this NLP project. In the first place, the t-davidson data only required the removal of leftover HTML elements. It did not require any kind of imputation or scaling. The /pol/ dataset did have issues of missing data, but this was only relevant for the features that contain the replies and the country names. When a post is empty, it would be senseless to come up with a comment of your own. When the country name is lacking (due to a non-geographical flag being used by a commenter), it would be unfair to attribute the related comment to any specific country. For this reason, there was no choice but to remove rows where either of that data was missing.

To satisfy this part of the document, operations have been performed on alternate datasets to show various imputation and scaling methods. Each of the topics will be handled separately.

There are many different imputation techniques such as:

* Mean substitution
* Mode substitution
* Hot deck imputation
* K-Nearest Neighbours (KNN) Imputation
* Interpolation

This list is not comprehensive and there are many more imputation techniques to research. Some of these listed methods will be explored, utilized, and further expanded upon to discuss their pros and cons.

The imputation examples will utilize a random dataset about travel times, taken from Kaggle. After downloading the dataset, it can be loaded into Jupyter Notebook using familiar Pandas functions. This is done as shown in figure 42.



*Figure 42: Loading of the travel times dataset*

The optional arguments are given to take the ”Date” and “StartTime” features of the dataset and convert them into an index that shows both the date and the time for each of the rows. A head() function can be used again to see what the data looks like. This is shown in figure 43.

A picture containing table

Description automatically generated

*Figure 43: First 5 rows of the travel times dataset*

The index has been correctly set. This step was necessary for another imputation method that will be utilized later. It’s visible that there is missing data in this dataset, such as partially in the “MaxSpeed” feature. The “FuelEconomy” and “Comments” features also appear to contain missing values. The number of missing values for each feature can be inspected as shown in figure 44.

A picture containing table

Description automatically generated

*Figure 44: Missing values in the travel times dataset*

The first imputation techniques to experiment with are mean and mode substitution. These methods are rather self-explanatory, and it means that missing values in a column will be replaced with either the mean of that column’s values, or the mode of that column’s values. These are very simple imputation methods, making them sufficient for when your data is normally distributed. However, when other components are present in the dataset such as a trend, then these methods will not suffice. Having a very odd distribution of value may also make the mean less suitable for use.

To use these methods (and more), the SimpleImputer function from scikit-learn can be utilized. This imputation function allows for a strategy to be selected, which gives the following possibilities:

* Mean
* Median
* Most frequent (mode)
* Constant

The mean imputation can be implemented as shown in figure 45.

Graphical user interface, text

Description automatically generated

*Figure 45: Mean imputation example*

A copy of the DataFrame is first made, and the imputation function is used to transform missing values into the mean of the ”MaxSpeed” feature’s values. Afterwards, the figure can be plotted, using “AvgSpeed” as the other axis since it’s known that it does not contain any missing data. The green points are transformed data and blue points are original non-missing data. The scatter plot appears converge on the middle of the plot, where most of the transformed data points lie as well. This makes mean imputation a decent option for this specific scenario.

The same code can also be used for the mode substitution by changing the strategy argument to most\_frequent. The implementation of this code is shown in figure 46.

Graphical user interface

Description automatically generated

*Figure 46: Mode imputation example*

The results seem to be very similar to that of mean imputation, with minor differences if observed carefully. Thus, this seems to be a well-performing method of imputation for this scenario as well. However, it’s important to always consider the negative effects imputation methods may have. Mean imputation can introduce bias into the standard error, and if values are not missing at random it can also introduce bias into the actual mean/mode of the column as well. Depending on the volume of missing data, imputing in this way can also affect the true relationships between columns. If a column has many outliers, then it may be preferred to use median imputation over mean imputation. In either case, either of these methods may be preferred over hot deck imputation. This method takes a random value present within a feature and fills that respective feature’s missing data with it. It’s a simple implementation but it may quickly introduce bias into the dataset.

During the loading of the dataset, it had been turned into a time series due the usage of dates and times to create the index. This is done to better illustrate the next imputation methods. It’s also possible to imputation methods known as forward fill and backwards fill. What these methods do is, when they encounter a missing value, is taking the next or the previous value and filling the missing value with it. These are both variants of interpolation. The chronological data of the “MaxSpeed” feature, without any imputation, is first shown in figure 47.

Graphical user interface

Description automatically generated

*Figure 47: Time series data of the “MaxSpeed” feature*

Implementation of the different filling methods is then performed as shown in figure 48.

Graphical user interface, chart

Description automatically generated

*Figure 48: Forwards and backwards filling of the “MaxSpeed” feature*

The plot shows the original blue data points in addition to the forward’s filled and backwards filled data points at the same time. The red data points represent the forward’s filled data, and the green data points represent the backwards filled data. Due to vertical position of the data points, it’s clearly observable that the red data points copy the value of the next data point, and the green data points copy the value of the previous data point. Both methods work, and they are decent solutions for patching up a time series like this.

However, there are better interpolation methods available as well. One such example is the interpolate() function from the Pandas library, whose implementation is shown in figure 49.

Graphical user interface, chart

Description automatically generated

*Figure 49: Interpolation of missing data points*

In this case, the choice has been made for a linear interpolation, which fills missing values with an increasing order between the previous and next observed values. There are various other arguments to give to this function as well, such as polynomial, spline, nearest, quadratic, etc. This means that interpolation is very customizable, but it’s clear that the created values are more natural than the ones created by forwards or backwards filling. As an example, the jump between data points at 2011-10-15 would be rather jarring without another value in between. This interpolation method also revealed that there were two missing data points in a row at the very end of the dataset. This type of interpolation also creates a more natural bridge across multiple missing data points in a row.

There are also more advanced methods of imputation, such as predicting them with K Nearest Neighbours. This is an algorithm used for simple classifications. The algorithm looks for features that are similar, to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set. This can be very useful in making predictions about the missing values. A method like this can be much more accurate than using a mean, median, mode, or other interpolation methods, but it’s also computationally more expensive. An example implementation is shown in figure 50.

Graphical user interface, text, application

Description automatically generated

*Figure 50: KNN imputation of missing data points*

Since KNN can only work with numerical data, it was necessary to subset some of the columns to leave out text data. In addition, since it’s a distance-based algorithm, it was necessary to scale the data to improve the usage of the model. Scaling will be more expanded upon in the next section. There are many parameters to tinker with for an implementation like this, but it’s not necessary to go through them all. There are many arguments to give to the KNN imputation function, but they have intentionally been omitted as this is beyond the scope of this explanation. By not supplying any, the function will rely on its default settings. Afterwards, both the existing values are once again plotted in blue, and the imputed values are plotted in green. They are more scattered as they contain different values, instead of all the same mean or mode.

KNN has several advantages like being easy to implement and the ability to work both on numeric and categorical data types. However, it can be tricky to define its number of nearest neighbours (which defaults to 5), as it introduces a trade-off between speed and accuracy. However, it always remains a viable method for imputing accurate values.

Scaling refers to the transformation of data to fit a specific scale, such as a scale of 1 to 10, or 1 to 100. Scaling is important for the usage of algorithm that rely on how far apart data points are, such as the K Nearest Neighbours implementation that was just utilized. Scaling variables will help with comparing things in an even playing field. There are different methods of scaling as well, such as:

* Standardization
* Minmax scaling
* Absolute maximum scaling

The standard scaling is calculated by subtracting the mean and dividing by the standard deviation. The minmax scaling is calculated by subtracting the minim value and then dividing by the difference between the maximum and minimum values. Absolute maximum scaling functions by finding the absolute maximum value of a feature in the dataset and dividing all the values in that feature by that maximum value. Each of these methods is available through the pre-processing section of the scikit-learn library. They can be imported and created as shown in figure 51.

Text

Description automatically generated

*Figure 51: Importation and creation of the different scalers*

For this scenario, the “AvgSpeed” feature from the travel times dataset will be used to ensure that no missing values will potentially mess with the scalers that are just about to be used. Next, all the scaling can be done at once by using the code in figure 52.

Graphical user interface, text, application

Description automatically generated

*Figure 52: Applying all scaling methods and saving the results in their respective variables*

First, the original data can be observed by creating a simple plot of the “AvgSpeed” feature. This is done as shown in figure 53.

Graphical user interface, chart, line chart

Description automatically generated

*Figure 53: The original “AvgSpeed” feature of the travel times dataset*

It’s seen that the values within this column range between about 40 and 110. In addition, the dataset appears to be quite spiky at times, which is the result from large differences between values of time periods. These large differences could potentially throw off machine learning models that look these distances between values, creating subpar results. Now the scaled results can be observed, of which the first one will be the results of the standard scaler. This is shown in figure 54.

Chart

Description automatically generated with medium confidence

*Figure 54: Results of the standard scaler*

The standard scaler makes sure that all the features are centred around the mean value with a standard deviation value of 1. In this case, the standard scaler has created a scale of -3 to 3. Overall, the results look much more fluid, and while visually large spikes still occur, the distance between -3 and 3 is much smaller compared to 40 and 110. Next, the minmax scaler and the absolute maximum scalers can be looked at, which are visible in figure 55.

Chart, line chart

Description automatically generated

Graphical user interface

Description automatically generated

*Figure 55: Results of the minmax scaler and the absolute maximum scaler*

It’s worth noting that all these results appear to have the exact same shape. If you were to overlap them in an image editing program and adjust opacity, there would be no difference. However, this is only visually as their scales do differ. This shows that, due to the simplicity of the data, there is a clear image of how best to scale the, with the implementation of the ranges differing. Whereas the standard scaler has opted for a range that starts a negative value, these two scalers have decided to stay within positive territory.

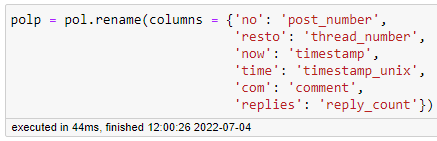
The minmax scaler ranges from 0 to 1, and the absolute maximum scaler ranges from about 0.3 to 1. It’s important to keep in mind that each of these scalers are simply mathematical calculations and it’s possible for entirely different ranges to be returned when used on a different dataset. What kind of model will be utilized will decide what kind of range is best used, and what kind of scalar would be the best choice to go for.

Overall, these results show that the scaling works, and that it’s a viable method of adjusting the data into a more suitable format for models that care about distance between data points. It can also be used to the benefit of many more Data Science-related implementations such as tree-based machine learning algorithms, linear and logistic regressions, supper vector machines, etc.

It’s now also clear why these types of imputation and scaling have not been necessary for the NLP implementation of this project.

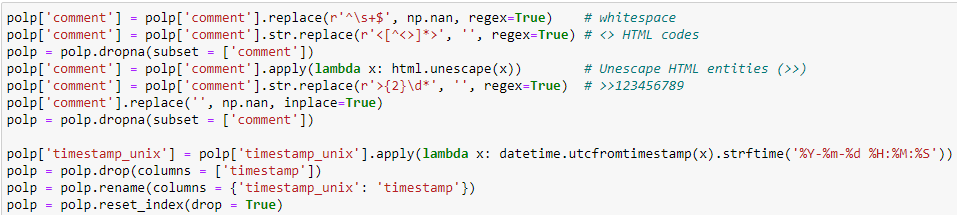
**2.4 You construct data by one-hot-encoding, defining targets & labelling relevant data**

After the exploration of both the /pol/ and t-davidson datasets, it was necessary to execute various minor cleaning steps to improve their condition. One of the first steps was the renaming of columns for the /pol/ dataset, which is shown in figure 46.



*Figure 46: Renaming of columns for the /pol/ dataset*

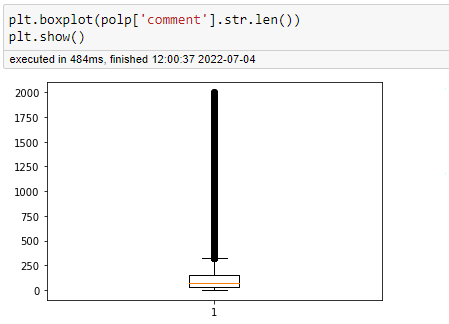
The original feature names weren’t very descriptive, which needed a change for the sake of legibility. Without proper naming, it would be very confusing to keep track of what each feature means exactly. Next, it was also necessary to deal with missing values and empty comments, which proved to be a hardy problem that required many operations. It was also necessary to decide upon which column to use for the time. All the performed operations for the /pol/ dataset are shown in figure 47.



*Figure 47: Cleaning operations for the /pol/ dataset*

The very first line uses regex to replace any comments containing only whitespace with proper nan values. The second line uses regex to deal with HTML tags which are present in the API’s results. These are the opening and closing tags used for the construction of websites. Afterwards, it’s necessary to drop the rows that currently have missing values in the “comment” feature, or the following operation would not work. In addition, the operation to unescape remaining HTML entities (such as &nbsp) required the application of a lambda in an apply() function, or it would throw various errors. The line after takes care of how users on 4chan boards reply and quote others using right-facing arrows and the chosen post numbers (e.g., >>123456789). With everything possibly unnecessary changed to nan values, all the rows with missing values in the “comment” feature are once again dropped.

It was also decided that the UNIX timestamp feature would be selected to serve as the feature to contain time period values. Due to being able to convert them into timestamp values, it’s guaranteed that they will contain the correct formatting. Afterwards, the other “timestamp” feature is dropped, and the UNIX timestamp feature usurps its name. As lots of rows have been dropped at this point, it’s also necessary to reset the index.

A choice was also made to drop further rows based on the length of comments. The boxplot in figure 48 reveals that some of them contain a very high number of characters. 

*Figure 48: Boxplot of comment character length*

Such high numbers of character translated to many words, which means many tokens to be made during the tokenization process for the NLP checkpoint. Due to the requirement of tensors being rectangular, all the smaller sequences would have to be padded to extreme lengths to match the longest sequences. To pre-emptively relieve stress from the checkpoint that will be used in the future, it has been decided to only use comments with less than 500 characters. The code for this operation is shown in figure 49

Graphical user interface, text, application

Description automatically generated

*Figure 49: Keeping only comments less than 500 characters long*

The t-davidson dataset also required a few operations to improve its quality, which are shown in figure 50.



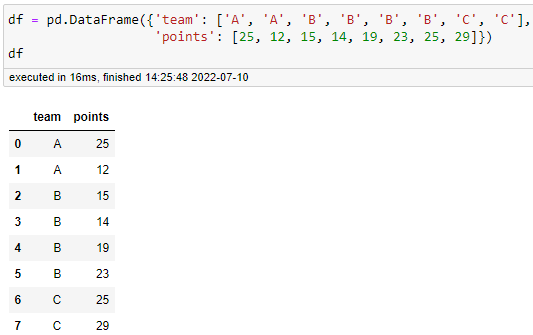
*Figure 50: Cleaning operations for the t-davidson dataset*

It was noted that the meanings for two of the labels were switched around in this dataset. To ensure that each label corresponds to the correct meaning as originally intended, they are swapped with the first three lines of code. The first 4 columns are then dropped as they were deemed unnecessary for the entire modelling process. HTML elements are also unescaped back into punctuation. Finally, the two remaining features are renamed to better fit the context of the project.

With properly cleaned datasets available, it’s now possible to talk about this chapter’s topic. As the t-davidson dataset is the one that will be given to the fine-tuning process of the HateXplain checkpoint, this will be the dataset to focus on for the first topic. As of now, there are only 2 features remaining, making it simple to determine what the dependent and independent variables are. The dependent variable (also known as the target variable) is the variable which holds the phenomenon that is being studied. In the context of this project, it’s the “label” feature, as it corresponds to the types of speech we are trying to classify. The independent variables are other variables which are utilized to explain the effect or output of the dependent variable. In this case, it can only be the “comment” feature. This division makes sense as through the sequences given are we trying to figure out which speech type (label) to classify them as. It’s possible for there to be multiple independent variables that all contribute to the prediction of the target variable. Even if the /pol/ dataset were to be used for the fine-tuning process, only the user comments would still be needed.

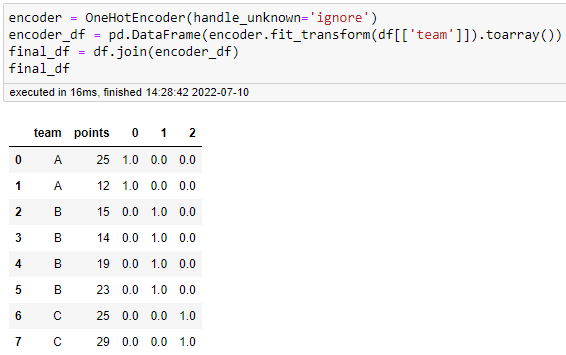
The concepts of one-hot-encoding and discretizing data have not really occurred for this project. It could be argued that the tokenization process is a form of one-hot-encoding as it transforms textual inputs into numerical equivalents, but this feels like a cheap argument to use. Just like the imputation and scaling subject, proof will be collected through the usage of example code and explanations.

One-hot-encoding is a method used to convert categorical variables into a format that can be used by machine learning algorithms. Just like the tokenization process, textual values are transformed into numbers that the algorithm can properly understand. The idea of one-hot-encoding is the creation of new variables that contain either 0 or 1 to represent the original categorical values. This is also like the padding and attention mask topics that are relevant to the tokenization process. An example DataFrame to perform one-hot-encoding on can be constructed as shown in figure 51.



*Figure 51: Example DataFrame to use for one-hot-encoding*

This example DataFrame contains 3 teams: A, B, and C, respectively, with several points assigned to them multiple times. The categorical variables of A, B, and C would have no meaning to a machine learning algorithm, requiring them to be converted. This can be done by importing the OneHotEncoder() function from sklearn. Afterwards, the encoder needs to be created, similarly to how the scalers from sklearn were created as well. All the necessary operations are then visible in figure 52.

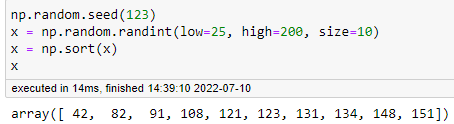


*Figure 52: Execution of one-hot-encoding*

The encoder is first created and stored in the encoder variable. There are various arguments that can be used by using the OneHotEncoder() function, but the default settings suffice in this simple example. The encoder is then used on the DataFrame, and the results are stored in the encoder\_df variable. Afterwards, the two DataFrames are combined to create the result. It’s visible in the figure that 3 new columns were created to accommodate the 3 unique values in the “team” column. The “0” column corresponds to team A as its contents contain a 1 for team A, and 0 for the other teams. The same is true for the columns “1” and “2”, and teams B and C, respectively. When this data is given to a machine learning algorithm, the algorithms can determine that the values containing 1 are valuable, and the values containing 0 do not. Of course, it’s also necessary to drop the original “team” columns and renaming the new columns to contain more descriptive names. For example, these columns can be renamed to “teamA”, “teamB”, and “teamC”, respectively. Choosing a name that correctly represents the original values is a smart choice to make to prevent future confusion. In this case, only 3 new columns were created, but it’s possible that this process creates many more columns in other contexts. The more columns there are, the more clarity is required.

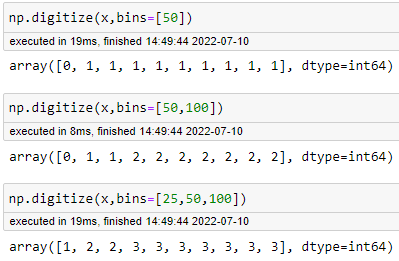
The topic of discretizing is about quantitative variables that you might want to bin or categorize depending on some on logic. An example is categorizing heights to either 0 or 1, with either of the labels containing a certain meaning. This could mean tall or short, above average, or below average, etc. This is possible to achieve through NumPy, Pandas, and various other methods.

An example array of values to discretize can be created through the usage of some NumPy functions. This method is shown in figure 53.



*Figure 53: Creation of a random set of numbers using NumPy*

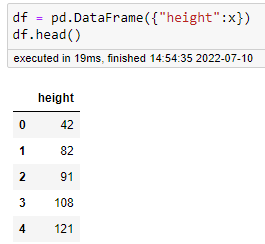
To utilize the random function, a seed first needs to be set. This can be anything, but in this case a simple value of 123 will suffice. As computers cannot truly perform randomness, it’s necessary to use a seed to determine the random output values. Even though functions to create something random are called multiple times, using the same seed will mean that the same values are returned every time. Something that changes frequently is usually used for seeding, such as the time of day. A random set of numbers is then created ranging from 25 to 200, with a size argument set to 10 to ensure that there is some variety. They’re also sorted from low to high. 3 Variations of NumPy’s digitize() function are then called to illustrate how discretizing works, as shown in figure 54.



*Figure 54: Discretizing with NumPy examples*

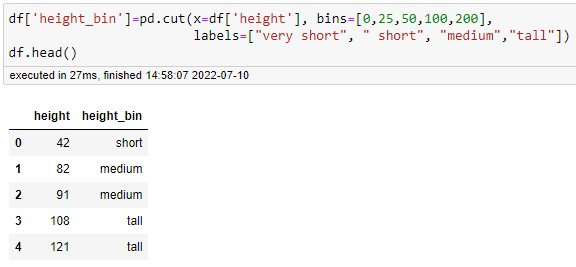
The first example uses only 2 bins, divided among lower or higher than 50. The results then show an array containing a single 0, and many 1s. What this function did is categorize the data into two bins depending on whether each data point is lower or higher than 50. As only 48 was lower, it’s given category 0, and the rest is given category 1. The other two examples show that it’s possible to keep creating more bins, which the function then divides accordingly. The higher the number of bins, the higher the numerical equivalents can go as well. The last example uses 4 total bins. There are no results with the number 0 as those are equivalent to being lower than 25. The single 1 then correspond to values lower than 50. All the 2s correspond to values lower than 100, and the 3s are bigger than the final bin of 100.

It’s also possible to perform discretizing with Pandas. This is done by first converting the variable x into a DataFrame format, as shown in figure 55.



*Figure 55: Transformation of variable x into a DataFrame*

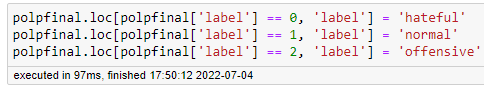
As we now have a DataFrame, it’s possible to perform all kinds of operations on it to create new columns. An example of discretization using Pandas is visible in figure 56.



*Figure 56: Discretization with Pandas example*

In this case, the values of x are distributed based on the bins of 0, 25, 50, 100, and 200. It’s also necessary to supply 0 in the bins when using this method. In addition, instead of using numbers, label names are supplied. In this case, the labels “very short”, “short”, “medium”, and “tall” have been chosen to represent each of the bins. Supplying meaningful label names is useful for being able to recognize what each category means.

While one-hot-encoding and discretization have not really occurred during the NLP project, a choice for labels did have to be made. The fine-tuning process returned the labels 0, 1, and 2, each with their respective meanings. After the labels were merged with the /pol/ dataset, they were attempted to be used within Power BI. However, Power BI refuses to create counts of rows containing the value 0 for some reason. The exact reason is unknown but counting the rows containing either 1 or 2worked as expected. For this reason, it was necessary to convert the labels of 0, 1, and 2 into proper, categorical names that could be counted within the program. While discretizing does not come into play here, a deliberate choice about names was still made. Figure 57 shows how the values were renamed.



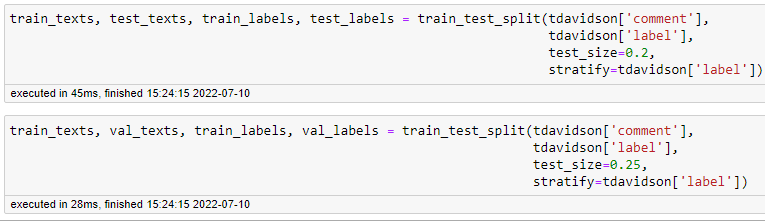
*Figure 57: Renaming of the 0, 1, and 2 labels in the final dataset*

The choice of names did not require much effort as the relations between the numbers and their meanings has been clarified many times before. Still, decisions were made to keep the values in lowercase, to represent how all the sequences were turned into lowercase versions as well. In addition, the words have been given the same format of being an adjective that could be prefixed to the word “speech”.

**2.5 You convert data formats as prerequisite for relevant model technique(s)**

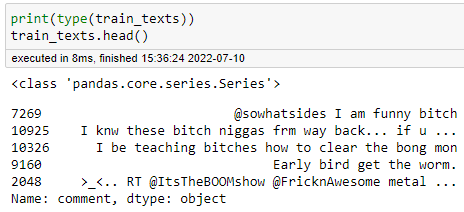
During the data preparation phase of the CRISP-DM cycle, the data underwent several transformations to accommodate the requirements of the checkpoint at various stages. However, all these transformations have taken place after the creation of the training, validation, and testing datasets. Chronologically, this is done in chapter 3.2, and these steps ae briefly touched upon there as well. However, explanations of the actual code implementations will be left to chapter 3.2, and this chapter will instead focus more on the transformations that were performed and the theory behind why this was done.

The t-davidson dataset started out as a CSV file that was loaded into Jupyter Notebook and turned into a DataFrame. A DataFrame itself is merely a collection of Series, not mattering much which data type or format it has exactly. Next, the data was divided into a training, validation, and testing set, done by using the code from figure 58.



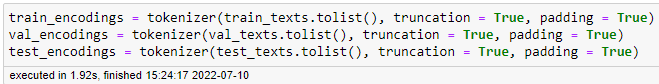
*Figure 58: Creation of the training, validation, and testing sets based on the t-davidson dataset*

Figure 59 then shows the data type and format of the resulting variables.



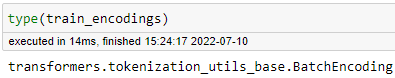
*Figure 59: Data type and format of the training set, and others*

As mentioned, a Series is simply one part of which a DataFrame is composed of. This means whether these results are a DataFrame or a single series does not matter, as a DataFrame can contain a single series. It would only slightly change the code needed to use each of these sets of data. However, the next step is very much important. After the 3 sets have been created, they need to undergo the tokenization process by using the HateXplain checkpoint’s tokenizer. This is done according to the code in figure 60.



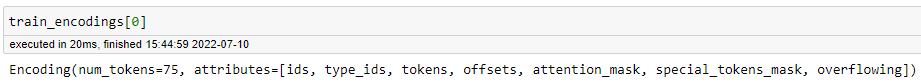
*Figure 60: Tokenization of the three sets of data*

The tokenization process created the encodings of the textual sequences necessary for the fine-tuning process. The data type of these encodings can be seen in figure 61.



*Figure 61: Data type of the encoding variables*

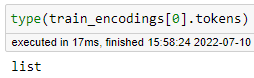
The data type has a rather lengthy name, but it can be dissected to better understand its meaning. First, the result originates from a method found in the transformers library, which is why it’s prefixed with transformers before the first period. The middle part refers to the base class common to both slow and fast tokenization classes. The final part is one of these tokenization classes: BatchEncoding which indicates that multiple sequences have been supplied to be tokenized at once. These results can also be further explored by peeking at one of the encodings, as shown in figure 62.



*Figure 62: Exploration of one of the encodings*

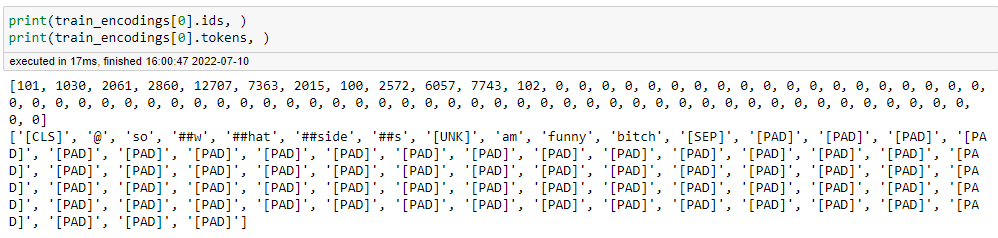
The encoding itself has the format of a Python dictionary, containing one field stating the number of tokens in this encoding, as well as plenty of attributes that can be explored as well.

Figure 63 shows how each of these attributes within the Encoding is a list.



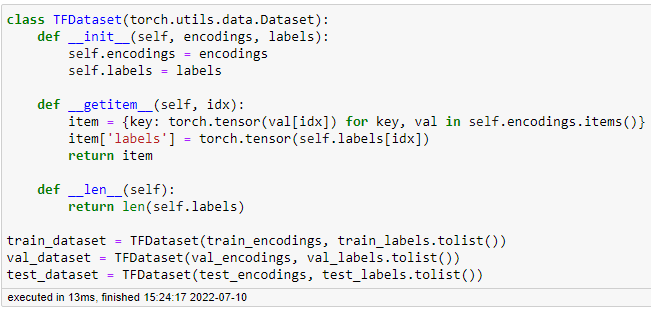
*Figure 63: Data type of an attribute within the encoding*

By looking at the ids and tokens attributes, it’s also possible to see numerical IDs and the tokens that were created, which will eventually be used for the checkpoint. These values are visible within figure 64.



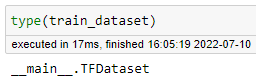
*Figure 64: Contents of the first index of the training set’s encodings*

The IDs show the numerical representations that were created, based on the dictionary of words that the tokenizer is familiar with. The tokens attribute shows how a sequence was split up into tokens to then convert into their numerical equivalents. As tensors needs to be rectangular, lots of padding tokens, represented by 0s in the IDs, are necessary to accommodate the longest input sequence. This is explained more in chapter 3.2. Before the fine-tuning of the HateXplain checkpoint can begin, one more transformation is necessary, which is shown in figure 65.



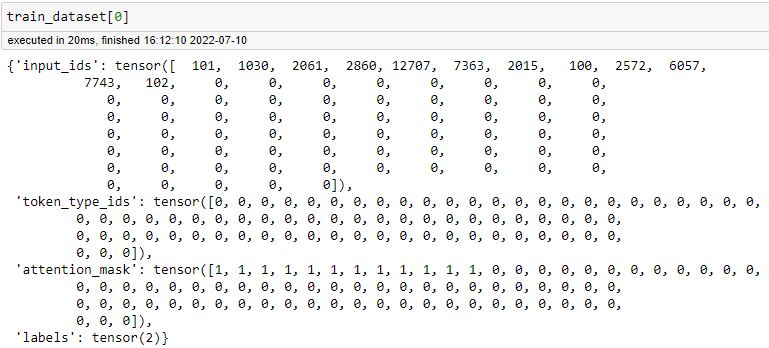
*Figure 65: Transformation of encodings and labels into a Dataset object*

This is a complicated piece of code that uses PyTorch to combine the encodings and the accompanying labels into one Dataset object. This is an abstract class representing a Dataset. This type of object is a map-style dataset that implements \_\_getitem\_\_() and \_\_len\_\_() to represent a map from indices/keys to data samples. The data type of the results can also be explored again, as shown in figure 66.



*Figure 66: Data type of the Dataset object*

This only returns the \_\_main\_\_ method of the TFDataset class name, which isn’t very useful. However, just like the encodings, positional arguments can be used to look at some of its contents. The results are visible in figure 67.



*Figure 67: Contents of the first index of the train\_dataset Dataset object*

It’s visible that this Dataset object has a dictionary or map-like structure, with many lists inside. Each of these lists is explained more in chapter 3.2. What matters here is the format of these lists, which are shown to be tensors. A tensor, originating from TensorFlow, is a generalization of vectors and matrices, better understood as a multi-dimensional array. This Dataset object containing its many tensors is the final, necessary format for the fine-tuning of the HateXplain checkpoint.

The trainer for the fine-tuning of the HateXplain dataset is expecting all this information to function properly. The input\_ids tensor contains the numerical representations of the tokens that were shown earlier. The token\_type\_ids tensor is used to show where sentences begin and end, when multiple are present in a sequence. In the shown example, only one sentence is present, making all values 0. The attention\_mask tensor contains information about which tokens need to be considered, and which ones can be ignored. This is a necessary component as many of the sequences in the batch are padded with [NULL] tokens (represented by 0 in the input\_ids and attention\_mask tensors0). This padding is necessary to ensure that tensors are rectangular, meaning that each sequence within has an equal length. Without, the tensor becomes unusable. The labels tensor shows the given label for this specific sequence.

This is the bundle of data that the trainer for fine-tuning expects to be able to work with. As such, the many transformations are necessary to accommodate these wishes. Without the creation of this exact Dataset object with the mentioned Tensor lists inside, the fine-tuning of the dataset would not work properly. For this reason, it has also not been possible to consider multiple different variants of data to use for the fine-tuning or modelling processes. The initial input is too simple (only text sequences and labels are required), and this preparation for the fine-tuning required a rigid routine that should not be tampered with. Note that this chapter of the document only tried to expand on why these transformations were necessary, and a more detailed explanation of the code implementations is given in chapter 3.2.

Learning Objective 3: You perform data analysis

**3.1 You split data into test & train sets to generate a test design**

Before any modelling can be done, it’s necessary to split your data up into multiple parts. These are referred to as the train, test, and optionally a validation dataset. Within the context of NLP, the model is trained on the training dataset, the validation dataset is given to see how the training operations of the model are performing, and the test dataset is used later to compare actual values and predicted values. As the HateXplain model is pre-trained, this method has already been performed on it by its creators. However, the fine-tuning of the checkpoint will be necessary to improve the model and these same operations will once again need to be performed.

The Hugging Face website not only contains models to use, but it contains datasets as well. Depending on how the creator designed these datasets, it’s possible that they have already been split up into the mentioned parts. However, the t-davidson dataset is not found on Hugging Face and was instead taken from a GitHub repository. This means that the process needs to be performed manually, and it is done as shown in figure 68.

Graphical user interface, text, application

Description automatically generated

*Figure 68: Splitting of the t-davidson data into a training, testing, and validation sets*

By using the train\_test\_split() method from sklearn, it’s possible to create all the datasets and store them in the right variables. Note that the train\_test\_split() function is being used on both the “comment” and “label” features of the dataset. This is to separate these comments and the labels as the they will need to be supplied separately to the trainer that will be utilized later. The train\_test\_split() function is used twice, and it follows a test design as shown in figure 69

A picture containing timeline

Description automatically generated

*Figure 69: Test design for splitting of the t-davidson data into a training, testing, and validation sets*

In addition to the training and testing sets, a validation set can be created as well. A validation set is optional, but it will be utilized in the upcoming trainer function, which will be utilizing epochs. An epoch indicates the number of passes of the entire training dataset the machine learning algorithm has completed. The validation set is used to fine-tune the checkpoint after each epoch. After the training phase is completed, the testing set will inform about the final accuracy that the model has been able to achieve.

According to Draelos (2019), common ratios of percentual division between the training, testing, and validation sets are:

* 70% training, 15% validation, 15% testing
* 80% training, 10% validation, 10% testing
* 60% training, 20% validation, 20% testing

In the shown code of figure 68, data is first separated into a training and testing dataset, with an 80:20 ratio. Afterwards, about 25% of the training data is used to create the validation dataset. In total, this creates a 60:20:20 ratio between the three sections. This approach is also taken to avoid overfitting. Overfitting is a common issue that occurs when a model performs well on training data, but poorly on unseen data. Detecting overfitting is very hard to do before the data has been tested, and only through trial and error will it really become clear whether your model is overfit, underfit, or just right.

Aside from the data to split and the split size, these functions feature a stratify argument as well. Data is usually chosen at random when splitting by using this function, which is fine. However, within the context of a classification problem, it would be an issue if one of the resulting splits does not contain some of the labels. The stratify argument can be used for these classification problems, which ensures that the resulting splits will always contain all the possible labels. For this use case, there are three unique labels.

It’s possible to confirm whether all labels are present in the resulting variables by using the code shown in figure 70.

Text

Description automatically generated

*Figure 71: Unique label count after usage of stratified splitting of the data*

Now that the splitting of the comments and labels has been successfully performed, it’s possible to continue with the required steps to prepare the model itself.

**3.2 You build & train relevant model technique(s) and create predictions using the model technique(s) on test data set**

The modelling has many complicated parts to it which require ample explanation. Before anything can be done, many functions from the various Hugging Face libraries need to be imported. This is done as seen in the following figure 72:

Text

Description automatically generated

*Figure 72: The import of various Hugging Face libraries’ functions*

First, the checkpoint and the tokenizer need to be imported which is done according to the code in figure 73.

Graphical user interface, text

Description automatically generated

*Figure 73: Creating the checkpoint and tokenizer*

First, the Hate-speech-CNERG/bert-base-uncased-hatexplain is saved as a string and given to the checkpoint variable. This is completely optional, and it can be given directly to the other functions, but I prefer this method for clarity. The tokenizer is then created by using the .from\_pretrained() function from the AutoTokenizer class, using the checkpoint as its argument. This will automatically fetch the data associated with the model’s tokenizer and cache it (so it’s only downloaded the first time you run the code below). As previously mentioned, these models from Hugging Face have already been trained before, which includes having to go through the process of tokenization. To ensure that further tokenization processes produce the exact same tokens for any words, it’s necessary to use the exact same tokenizer.

The model variable will be given the checkpoint, and it makes use of the .from\_pretrained() function of the AutoModel… class. There are many different AutoModel classes, usable depending on the task that needs to be performed. In this scenario, sequence classification is the relevant task, which means that the AutoModelForSequenceClassificationClass will be utilized. It’s also important to note that the terms text classification and sequence classification can be used interchangeably. If the .from\_pretrained() function of this class were to be called without providing any arguments, it would default to a standard sequence classification checkpoint. In this case, since the checkpoint is given directly, it will fetch the checkpoint from the Hugging Face hub.

It’s also possible to look at the checkpoint’s configuration to see what each of its labels mean, which is done with the code from figure 74.

Graphical user interface, text, application

Description automatically generated

*Figure 74: Labels from the checkpoint’s configuration*

As visible, these labels match the ones that have been expected so far. With the training, testing, and validation sets having already been created, it’s possible to continue with the next required steps. The tokenizer function is called to transform the training, testing, and validation sets into their numerary counterparts, using the checkpoint’s tokenizer. This is done as shown in figure 75.

Graphical user interface, text

Description automatically generated

*Figure 75: Tokenization of the training, testing, and validation datasets*

There are additional parameters where are relevant here, which are truncation and padding. Truncation will ensure that the maximum allowed number of tokens by the checkpoint is upheld. A custom value can be given, but if simply set to True then it will rely on the checkpoint’s maximum value. This will truncate token by token, removing a token from the longest sequence in the pair until the proper length is reached. The padding argument is also necessary to set to true due to how batch tokenization works. Batched inputs are often different lengths, so they can’t be converted to fixed-size tensors. Informally put, tensors need to be “rectangular”, meaning that the length of each input needs to be the same. When creating the sequences of tokens, padding adds special token characters (usually [0] to indicate whitespace) to ensure that all inputs are of the same length. However, these whitespace tokens need to not be ignored by the checkpoint as they might otherwise influence the predictions.

This is where the attention mask comes into play. The attention mask is a tensor which is created with the exact same shape as the input IDs tensor. This tensor is filled with 0s and 1s to indicate which corresponding tokens need to be ignored or not. 0 indicates that a token should not be attended to, and 1 indicates that the token should be attended to. An example of this attention mask is shown in figure 76.

Graphical user interface, text, application

Description automatically generated

*Figure 76: Example of padding and the attention mask*

It’s also possible to decode the resulting encodings to see how the tokenizer has affected the given input, as shown in figure 77, using the first entry of the train\_encodings variable as input.

Graphical user interface, text, application

Description automatically generated

*Figure 77: Example of a decoded token sequence*

The example results in figure 77 shows how few parts of the input are unrecognized and turned into [UNK] tokens. The string is also confined by the [CLS] and [SEP] tokens to indicate the start and end of the input. After that, many [PAD] tokens have been added to ensure that the tensor has a rectangular form. The unknown tokens at the beginning likely stem from completely unknown character sequences and user mentions that may occur on Twitter (where this data was gathered). However, the tokenizer picks up on the important, context-giving words in the sequence.

The train\_encodings, val\_encodings, and test\_encodings variables now contain the dictionaries of tokens for each of their respective input data. These variables are of the transformers.tokenization\_utils\_base.BatchEncoding data type, which need to be further transformed to become usable by the checkpoint. How this is done depends on whether PyTorch or Tensorflow is being used for the implementation. In PyTorch, this is done by subclassing a torch.utils.data.Dataset object and implementing \_\_len\_\_ and \_\_getitem\_\_. This is also necessary to glue the encoded input and the corresponding labels together. The implementation is visible within figure 78.

Text, letter

Description automatically generated

*Figure 78: Creation of the TFDataset Dataset object*

After this code has been utilized, the datasets have been sufficiently prepared for usage with Hugging Face’s trainer. Using the trainer to fine-tune the checkpoint requires multiple parts with many parameters, as visible in figure 79.

Text, letter

Description automatically generated

*Figure 79: Trainer and trainer arguments for fine-tuning of the data*

The trainer itself takes the chosen model (checkpoint), the training arguments, the training dataset, and the evaluation dataset as parameters. These are rather self-explanatory, except for the training arguments. These are created by calling the TrainingArguments class and instantiating various parameters within.

The output\_dir refers to the desired output directory where the fine-tuned checkpoint will need to be saved. The trainer does not adjust the original checkpoint, but it instead saves an improved version to the specified location instead. This will allow both versions of the checkpoint to be used for comparisons.

The num\_train\_epochs refers to the number of epochs to perform. As previously explained, this is how many times the entire training dataset will be used for fine-tuning in the model. In this scenario, the training will undergo 3 epochs.

The per\_device\_train\_batch\_size refers to the batch size per GPU/TPU core/CPU for training. In this case, 16 sequences are used at the same time. It’s an optional argument that defaults to 8.

The per\_device\_eval\_batch\_size refers to the batch size per GPU/TPU core/CPU for evaluation. In this case, 64 sequences are used at the same time. It’s also an optional argument that defaults to 8.

The warmup\_steps refers to how many updates with a low learning rate are performed before the training uses its default learning rate. The idea is that this helps your network to slowly adapt to the data intuitively. The learning rate is a parameter that determines how much an updating step influences the current value of the weights. The weights, in turn, determine how much influence certain input has over the output.

The weight\_decay is a regularization technique that causes the weights to exponentially decay to zero by adding a small penalty over time. This reduces the impact of certain inputs over time, and it is also used to prevent overfitting.

The logging\_dir refers to the directory where logs of the fine-tuning are kept.

The logging\_steps refers to how often logs are saved. In this case it is done every 10 steps during the fine-tuning.

In total, the TrainingArguments class has over 50 arguments to change around, making it is a large and difficult task to determine which arguments are good to tune. The currently chosen arguments have been determined based on examples given in documentation of Hugging Face’s trainer and trainer-related methods.

After everything is set up, the trainer can be executed by calling trainer.train(). After doing so, small warnings and more information are given immediately, as visible in figure 80.

Text

Description automatically generated with medium confidence

*Figure 80: Warnings and training information*

The warnings are simply about some upcoming changes in newer versions of the classes, which can be safely ignored for now. Afterwards, some information is given about the fine-tuning that is about to be performed. There are 18587 examples to train on, which is equal to the input of the training dataset. The epochs have been set to 3, which is how many times the training will go over the 18587 input sequences. The batch size has been set to 16 as specified. When training a model, the data is usually divided into mini-batches (16 in this case) and goes through them one by one. The model predicts batch labels, which are used to compute the loss with respect to the actual targets. Gradient accumulation modifies the last step of the training process. Instead of updating the model weights on every batch, we can save gradient values, proceed to the next batch, and add up the new gradients. The weight update is then done only after several batches have been processed by the model. Thus, gradient accumulation helps to imitate a larger batch size.

In total, the training will go through 3486 optimization steps. While training, the progress is also continuously updated, as shown in figure 81.

Graphical user interface, text, application

Description automatically generated

*Figure 81: Training loss per 10 training steps*

The training steps are shown, as well as the epochs and the time it will take to complete training.

After every 10 training steps, the training loss is shown. The training loss refers to how well the results at that step of the model fit the training data. A high loss refers to a high error, meaning that the results are inaccurate. In figure 81, it can be observed that the training loss starts out very high, which is normal as the model has yet to acclimate to the given input and figure out relations between input sequences and input labels. At later points during the training, the training loss keeps decreasing, indicating that the training process is improving, as visible in figure 82.

Table

Description automatically generated

*Figure 82: Gradual improvement in training loss during the training*

Every 500 training steps, a checkpoint folder is made and saved to the chosen logging directory, shown in figure 83.

Text

Description automatically generated

*Figure 83: Logging directory of the model fine-tuning*

Various files are contained within these folders, but the trainer\_state.json file can be observed to view the same values that are being logged live in the Jupyter Notebook environment. This is shown in the following figure 84.

Text

Description automatically generated

*Figure 84: Logged settings and training step results*

Files like these are saved every 500 steps and are useful for looking back on the progression of the training. The epoch, the learning\_rate, the training loss, the current step, etc. are all shown. Overall, the training performs smoothly when set up like this, without any issues arising during the process. The training also took a long time to finish (around 6 hours total). The amount of time required can be improved through various settings related to GPU usage, but I opted not to mess around too much with system settings and instead let the process run as is. After the fine-tuning has finished, it’s important to save the results to not lose any work. This is done as shown in figure 85.

Text, letter

Description automatically generated

Figure 85: Saving of the fine-tuned model.

The fine-tuning of the checkpoint has taken place on the training and validation datasets. To measure the performance of the model, the testing dataset will be used. The checkpoint has never seen this data during the training. After the splitting of the data, this part of the data contains 4,957 rows. It’s important to note here that for some reason, using the fine-tuned model is extremely memory intensive. Trying to use the fine-tuned checkpoint on about 500 sequence inputs quickly takes up more than 32GB of memory on my machine, causing the whole system to freeze. It’s unclear whether something is wrong with the fine-tuned model or whether it is simply this costly, but figure 86 shows the steady climb of memory being used upon execution.

Chart, line chart

Description automatically generated

*Figure 86: Memory usage of the fine-tuned checkpoint*

For this reason, the examples to determine accuracy of the models will be contained to about 250 of the 4,957 entries. To create these 250 sequences, the code in figure 87 is utilized.



Figure 87: Creating the 250 comments and labels of the testing dataset

First, the untrained checkpoint will be used. Several lines of code are necessary to guide the data through several transformations. Some of these are familiar by now, such as the usage of the tokenizer, but new concepts are relevant here as well. This code is shown in the following figure 88.

Text

Description automatically generated

*Figure 88: Usage of the untrained checkpoint*

To begin, the tokenizer is called on the testcomments variable, which holds the 250 sequences that will be used for this performance test. The padding and truncation are necessary here as well to create rectangular tensors. The return\_tensors argument is necessary here to ensure that the output is given in the form of tensors, as this format is necessary for the upcoming steps. In addition, it’s necessary to specify that they are PyTorch tensors, not Tensorflow tensors, due to the PyTorch framework being used for this implementation. Afterwards, this tensor is given to the checkpoint which was previously saved in the model variable. Once the checkpoint has worked its magic, output is given in the form of logits, which can be observed by looking at the input variable, as shown in figure 89.

Graphical user interface, text

Description automatically generated

*Figure 89: Output in logits after execution of the checkpoint*

Logits can mean various things, but within the context of neural networks it refers to the raw predictions that result from the final layer of a neural network. Without any further augmentation, these logits are completely useless. For this reason, PyTorch is used to apply a SoftMax over these logits. What this does is convert the raw logits to actual probabilities that can be read. The checkpoint reads a sequence and assigns a total probability of 100% over the 3 labels, depending on which labels it thinks fits each sequence the most. Also, the output is of the transformers.modeling\_outputs.SequenceClassifierOutput data type, which needs to be converted back into a tensor. The SoftMax function takes care of this as well. To make the results more legible and usable for further changes with Python and Pandas, I have opted to convert everything into a DataFrame. A way to directly change a tensor array into a DataFrame was not found, which is why the workaround was chosen to convert the tensor array into a NumPy array, and the NumPy array into a DataFrame. In addition, the probabilities are multiplied by 100 to get rid of decimals, and column names are given to show which probabilities relate to which predicted label. The results can then be observed as visible in figure 90.

Graphical user interface, text, application, table

Description automatically generated

*Figure 90: Classification results of the untrained checkpoint*

With this, the predictions of the model have become clear. However, a few more steps are necessary to utilize the chosen metrics, which will be handled in the next section.

**3.3 You assess the model technique(s) on chosen metrics of the defined success criteria**

Now that both the regular and fine-tuned model exist, it’s possible to compare their results. First needs to be determined how the results will be measured. Hugging Face has many metrics available to use, and which ones will be utilized has been defined beforehand. The load\_metric() method from the datasets library can be used to load the metrics from Hugging Face. This is done as shown in figure 91.

Text

Description automatically generated

Figure 91: Loading of the accuracy metric from Hugging Face

Due to the AUC ROC metric’s versatility, it’s important to supply the “multiclass” argument to ensure that the right version of the metric is loaded, since this specific NLP task works with 3 total labels to predict.

This data needs some more changes to be usable for some of the metrics. Naturally, the label with the highest probability, is what the checkpoint thinks each sequence is most likely to be. A new column is first made to store the chosen label with the highest value across each row in. This is done by using the code in figure 92.

Graphical user interface, text

Description automatically generated

*Figure 92: Creation of the new label column in the modelresults DataFrame*

By making use of idxmax() function, the new column is filled up with the column names of each row with the highest probability. After this, each result in this column can be changed into its corresponding numeric value according to the labels that have been defined at the very beginning. Recall that 0 corresponds with hate speech, 1 with normal speech, and 2 with offensive speech. This conversion is done with the code in figure 93.

A picture containing icon

Description automatically generated

*Figure 93: Conversion of label names to corresponding label numbers*

The results can be observed in the following figure 94.

Table

Description automatically generated

*Figure 94: Final look of the new label column in the modelresults DataFrame*

Finally, the previously loaded metrics can be utilized to see what kind of results the checkpoint has managed to achieve. The results of the metrics can be seen in figure 95.

Graphical user interface, application

Description automatically generated

*Figure 95: Metric results of the untrained checkpoint*

This accuracy reveals that the untrained checkpoint works decently and is by no means perfect. The checkpoint can be used for classification, but it’s likely that many of the results will be wrong. In addition, it can already be observed from figure 94 that the checkpoint isn’t very confident in its choice of labels.

For the F1 metric, it’s necessary to supply the average argument stating “macro” to ensure that the metric can perform its operations on multiple classes (labels in this case). The results show a low score, even lower than the accuracy of the checkpoint. This low F1 score means that the checkpoint seems to be making many wrong choices, and its accuracy is partially propped up by accidental right choices such as false positives.

Lastly, the ROC AUC metric can be looked at as well. As stated before, the score being close to 0.5 is the worst results of the ROC AUC score, as this would mean that classifications are happening as if it was performed by a coinflip. A score higher than 0.5 is good, and scores below 0.5 are worse. In this case, a score of about 0.84 has been reached, which indicates that the checkpoint seems to be making informed decisions. However, due to the low accuracy from the accuracy metric, it seems that these decisions are ill-informed. This means that further fine-tuning may alleviate these problems.

The fine-tuned checkpoint can now be imported in similar fashion to the untrained model. Instead of from Hugging Face, the fine-tuned checkpoint will originate from the location where it was saved on my PC. The code in figure 96 is used to load the checkpoint.



*Figure 96: Loading of the fine-tuned checkpoint*

Afterwards, all the same steps can be used in the exact same way as for the untrained model. To ensure clarity, all the relevant variables have been prefixed with trained- as well. The results of the utilization of the metrics for the fine-tuned model are shown in figure 97.

Graphical user interface, application

Description automatically generated

*Figure 97: Metric results of the fine-tuned checkpoint*

The results from the fine-tuned model show a massive increase in accuracy, reaching up to 98.4%. This means that not only are very few of the labels chosen wrongfully, but also that the fine-tuned version has far surpassed the original. In addition, the accuracy more than fulfils the data mining success criteria of reaching at least a 90% accuracy.

The F1 score is very high as well, albeit lower than both the accuracy and the ROC AUC score. This means that there are very few false positives and false negatives present during the decision-making point of labelling the data.

A very high ROC AUC score also shows that the labelling of the data is not left up to chance and that the model is very sure about its predictions. Due to how a high percentage of accuracy has been achieved as well, the fine-tuned checkpoint knows what the correct labels to choose are, and there is little to no doubt when making these decisions.

At this point, several of the data mining success criteria can be compared with the results so far to determine whether they have been achieved or not. These success criteria are:

* The NLP model has been fine-tuned on additional data
* The NLP model can classify between normal, offensive, and hateful speech
* The NLP model has a 90% + accuracy with its classifications

The first criterion required the checkpoint to be fine-tuned on additional data. This has been achieved with the help of the t-davidson dataset. It has also been implemented properly by making the dataset identical to the format that was used for the pre-training of the HateXplain checkpoint. This required some minor changes such as switching two label definitions around. Thus, this success criterion has been achieved.

The second criterion demanded a classification between 3 labels. This is how the HateXplain checkpoint functioned, and it was thus desired that any further implementation would also work this way. As the fine-tuning and utilization of the checkpoint has passed successfully, this criterion has also been achieved.

The third criterion required the acquisition of a high accuracy with the checkpoint’s classifications. This is a natural choice as an inaccurate or unsure model cannot be trusted. Bad to average results would require more research into possibilities of improving the checkpoint or choosing another one altogether. The results have shown the fine-tuned checkpoint to both very accurate and decisive.

Due to chronological order, the upcoming steps after the modelling are performed to implement and satisfy the remaining success criteria. These success criteria are:

* The custom /pol/ dataset has been classified using the NLP model
* The results of the classification can be viewed in a Power BI dashboard

The first is handled in chapter 2.2 of this document, and the second in chapter 4.3.

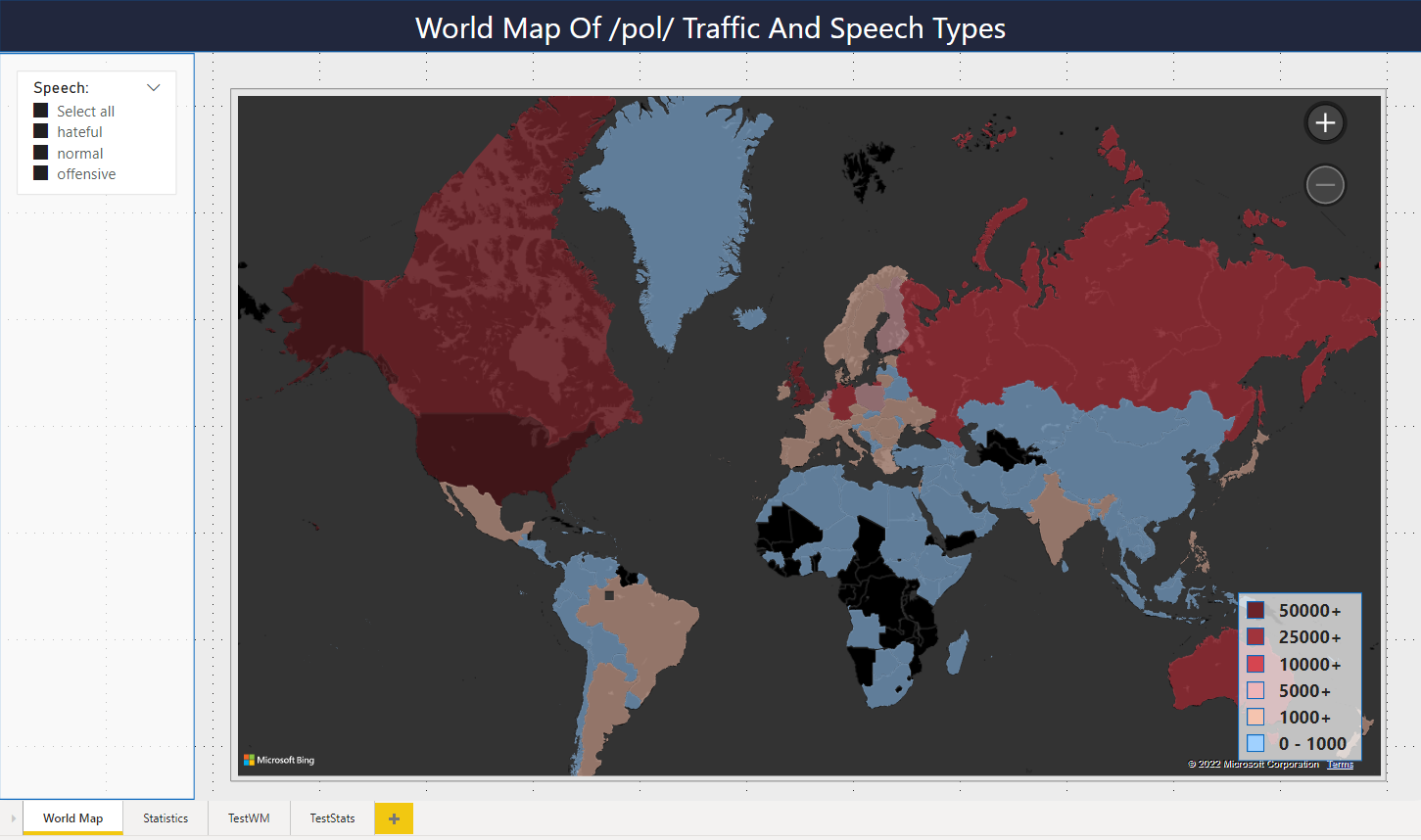
Learning Objective 4: You evaluate & deploy results of the Data Science process

**4.3 You produce a deliverable for the customer**

This chapter is intentionally done first to accommodate the final data mining success criteria of this project before its achievements will be judged. The business objectives stated that results of the project need to be visualized. Data mining goals and success criteria were created in such a way to allow for the creation of a Power BI dashboard wherein results can be visualized. Power BI has been chosen as the tool of choice due to familiarity and how it’s simple to create the desired visualisations through a click-and-drag method. In addition, all the elements are interactive in various ways, allowing for users to inspect specific parts of the visualisations further.

The NLP checkpoint, through its classification, has created statistics of how much normal, offensive, or hateful posts have been made over a period of about 4 days. This data does not encompass 4 full days exactly, and more data collection can be performed in the future to acquire longer periods of time. Thanks to insights gained from the modelling process and prior research related to Data Visualization, it’s known which visualizations will be useful to incorporate into the dashboard. Due to geographical location being available, it’s possible to create a choropleth map of message volume. It’s also possible to create bar charts and pie charts as multiple categories of speech types and countries are available. There is also a small time-component available for use, which allows for the creation of a line chart. While the line chart might not be too useful yet due to the small period of time, it can always be expanded upon in the future.

Recall that chapter 2.2 handled the merging of the /pol/ dataset and the created labels. Now that a final version of the dataset exists, it’s possible to load the CSV file into Power BI. The dashboard itself consists of two different pages; one for the choropleth map, and another for the other statistical visualizations. These two main pages are visible in figures 98 and 99.



*Figure 98: World Map page of the /pol/ dashboard*

Graphical user interface, chart

Description automatically generated

*Figure 99: Statistics page of the /pol/ dashboard*

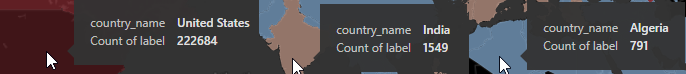
There are lots of different details to talk about in this dashboard. For the choropleth map, a custom colour scheme has been selected. The colour scheme ranges from blue to red, with manually defined thresholds to indicate higher volumes. This was a necessary procedure due to North America having a much higher volume than the other countries. Only North America would show up as red, and any other country would remain blue due to the difference. This custom implementation shows more variety. This range of colours and thresholds is shown in figure 100.

Graphical user interface

Description automatically generated with medium confidence

*Figure 100: Colour scheme settings for the choropleth map*

In terms of functionality, it’s possible to mouse over each individual country to see how many total messages have been sent from that geographical location. This is visible in the examples shown in figure 101.



*Figure 101: Examples of total message volumes per country on mouseover*

Note that the examples from figure 101 are the total message volumes, containing the sum of all 3 types of speech. This can be adjusted by using the checkboxes in the small menu on the left. In figure ##, all of them are currently selected, but they can be manually turned off or on, and the values inside of the choropleth map will update automatically. Figure 102 shows the values when only the “Hateful” label is selected.

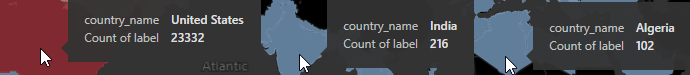


Figure 102: *Examples of hateful message volumes per country on mouseover*

It’s also possible to freely drag the map and zoom in or out by using either the mouse’s scroll wheel or by using the buttons in the top right of the choropleth map.

The second page has more visualizations and shows various statistics that were possible to make with the created data. The vertical visualization on the left shows the volumes of each speech type for the top 10 countries. It’s clear that the United States dominates in terms of total volume. Each of the bars is made up of 3 different parts that reflect the total values of each category of speech. The slider on the left-hand side can be used to zoom in and out on certain parts of the visualization to take a better look at some of the other countries. Figure 103 shows the visualization with the slider dragged down.

Chart, bar chart

Description automatically generated

*Figure 103: Zoomed in version of the speech type volumes of top 10 countries visualization*

The top right section of this dashboard’s page contains two visualizations that convey the same information in different formats. The distribution of speech types shows how most of the data is made up of normal speech. Offensive speech is the next largest category, and hateful speech is in last place. Despite the large volume of regular speech, the offensive and hateful speech categories combined comprise almost 30% of the total data. This means that about every third message posted contains language that ranges from slights and insults to prejudice and hate.

The final visualization in the bottom right gives an indication of the volume of speech per day. Data was collected from the board’s archive, which can only go so far into the past. Data collection was started at a time where unfortunately only very little data of the 23rd of May remained. That is the reason why the line chart starts off close to 0. Afterwards regular progression of the data point is visible, with the numbers dipping near the end of the graph again as that day had not fully passed yet. Even though not that many days’ worth of data is available, the graph is still able to convey that at least about 100,000 - 150,000 messages are posted daily. This type of chart will naturally be enhanced once more data will be collected.

Finally, it’s also possible to focus on certain types of speech by clicking on them. An example is when the United States’ 23,000 hateful messages section of the leftmost chart is selected. Figure 104 shows the behaviour of the entire dashboard when this happens.

Graphical user interface, chart, application

Description automatically generated

*Figure 104: Dashboard behaviour when focused on certain values*

It’s visible that all elements containing hateful speech have now been focused on, with others being made opaquer or omitted entirely. The total distribution of speech charts also shows for how much the American hateful speech counts of the total amount of hateful speech.

**4.1 You evaluate and match success criteria with business objectives of the Data Science process**

With the NLP project completed and with the dashboard made, it’s possible to judge whether all the objectives and success criteria have been met or not. To recap, the business goal had been defined as follows:

*“Utilize Data Science and machine learning to measure and visualize the prevalence of hate within the /pol/ community”*

The business goal’s success criteria were also defined as such:

* Distinctions are made between hateful and non-hateful speech
* The volume of each speech type is categorized and visualized

This goal and its success criteria were defined to align with the creation of a first Data Science-foray that will allow The Equal Web to garner a first understanding of a community such as /pol., and to pave a way for deeper research in the future. The goal is a broad pursuit, with its success criteria helping to focus on what is possible for now. There was an idea of what needed to be achieved, but there was no idea of what technologies could be utilized for this purpose. To create more concrete goals to work on during the Data Science process, related data mining goals and success criteria were created as well. To recap, the data mining goals had been defined as follows:

* Implement a supervised NLP model to classify different speech types
* Create an environment wherein the NLP model’s results can be displayed and reviewed

After research, it became clear that NLP would offer ample opportunity to provide the results that were necessary to satisfy these data mining goals. At this point, research was also done into which specific models and programs would be useful to look at. Based on the research and the data mining goals, the following success criteria were created:

* The NLP model has been fine-tuned on additional data
* The NLP model can classify between normal, offensive, and hateful speech
* The NLP model has a 90% + accuracy with its classifications
* The custom /pol/ dataset has been classified using the NLP model
* The results of the classification can be viewed in a Power BI dashboard

It was clear that NLP-related models could be used right out of the box, but that there would be no guarantee about their efficacy. The process of fine-tuning was recommend and necessary, not only for the collection of proof for this document, but also to ensure that the model can achieve a high accuracy. It also became clear that with the chosen HateXplain checkpoint, speech could not only be divided between hateful and non-hateful, but between a third category as well. For this reason, the data mining success criterion of classifying between 3 types of speech was created to accommodate this possibility. The classifications and predictions of models also need to be highly accurate and decisive, otherwise the results cannot be trusted. While it’s impossible to predict beforehand what kind of accuracy will be achieved, it’s necessary to strive for a high percentage such as 90%. This high accuracy is also needed to entrust the model with the /pol/ dataset, which has not been labelled beforehand. And finally, all the results needed to be visualized in a Power BI dashboard-the chosen tool of choice- to create an environment for the stakeholders to interact with.

Firstly, the NLP checkpoint has successfully been fine-tuned on additional data that was supplied in the form of the t-davidson dataset. By slightly altering the dataset so that the label definitions matched those of the HateXplain exactly, it was possible to execute the training process for the fine-tuning of the checkpoint. Before the fine-tuning, the checkpoint achieved the following scores for the chosen metrics:

* ACC: ~0.65
* F1: ~0.46
* ROC AUC: ~0.84

After the fine-tuning, the metric results changed to the following values:

* ACC: ~0.98
* F1: ~0.94
* ROC AUC: ~0.98

It was apparent that the regular version of the checkpoint achieved average results when used on the testing set of the t-davidson dataset. Average was not enough, however, and the fine-tuned results show what is possible with some extra training. The fine-tuned version is extremely accurate and decisive. Thus, the first success criterion has been achieved.

The checkpoint can also successfully decide between the 3 labels of normal, offensive, and hateful speech. Neither of the labels is being misrepresented, predicted wrong consistently, or is showing any other signs of negative impact. The checkpoint has also neither been overfit or underfit. Thus, the second success criterion has been achieved.

As shown, the checkpoint also achieved an accuracy of at least 98%, in addition to the stellar F1 and ROC AUC scores as well. Thus, the third success criterion has been achieved.

Thanks to how the fine-tuned checkpoint produced incredible results, it could be trusted with the classification of the /pol/ dataset as well. After a 17-hour long process, all 455,503 sequences were successfully classified. Thus, the fourth success criterion has been achieved.

Lastly, it was necessary to funnel the results into a Power BI dashboard wherein visualizations could be made to show the results of the Data Science project. The created dashboard features two separate pages: one with a choropleth map for a visual perspective of the data distribution between countries, and another with more statistical information and visualizations. Thus, this success criterion has been achieved as well.

Thanks to the achievement of these success criteria, it has been possible to implement an NLP checkpoint to classify different types of speech and create an environment wherein all the results of the NLP checkpoint can be displayed and reviewed. Thus, the data mining goals have been achieved as well.

The business success criteria demanded that speech be divided between hateful and non-hateful. The results of this project went beyond that thanks to the HateXplain checkpoint, and made the separation between the two desired classes, and an additional class as well. The volumes of each speech type have been categorized and visualized as well. In addition, they have also been divided on a country-basis, allowing for additional information to be gleamed in the dashboard. Thus, the business success criteria have been attained as well.

Through the accomplishments of all these various goals and success criteria, the overarching business goal has been satisfied as well. The project was fairly simple and humble in its execution compared to more complex situations, but it sufficed well as a good start to The Equal Web’s future plans.

Various insights have been collected during the project as well, especially since this was my first contact with NLP. I learned not to underestimate some of the NLP-related activities that take place. The fine-tuning of the HateXplain checkpoint took about 6 hours, and the labelling of the entire /pol/ dataset took about 17 hours. These issues occurred partly due to how the fine-tuning and classification processes did not utilize my GPU. There seem to be various options to optimize NLP processes through GPU-utilization that would speed everything up. The Hugging Face tutorials often linked to copies of their code examples in Google Colab. This environment offers the utilization of GPU-related improvements as well, but this didn’t seem available right away. Thus, it will be necessary in the future to investigate various settings related to these performance improvements.

There were also various points where I ran into memory issues, requiring the creation of workarounds with loops to avoid bricking my system. It is unknown as of now whether these NLP models are really that expensive memory-wise, or whether improvements could have been made to not overload my system’s memory. This is also a point where the process could have been improved.

In hindsight, it was also a bit of a gamble to depend on the achievement of a high accuracy percentage for the fine-tuned checkpoint. Due to it being unreasonable to label the /pol/ dataset by myself, it’s required to have a checkpoint that can be trusted to perform the labelling without messing up. If the fine-tuning did not correctly work out, it could have caused many problems for the project. This could also have been improved by looking into the subject more and coming up with alternatives for when it would be necessary.

**4.2 You determine next steps and set up an advisory report for follow-up**

At this point, all the goals and success criteria have been achieved, allowing for a satisfying conclusion to the project. However, this does not mean that there are no improvements to be made in the future. Especially due to how this project served as a first step on The Equal Web’s vast Data Science journey, there is much more research to be performed in the future. For this reason, several next steps can be created to cater to future projects. The following possible steps have been devised:

* Additional data collection
* Additional HateXplain functionality
* Additional NLP tasks
* Additional model selection

Right now, the data spans a period of close to 4 days, which does not allow for seeing user traffic and speech volume over periods such as weeks or months. The collection of additional data would satisfy the aspect of time that can be used for different projects. Once adequate data has been collected, it would even be possible to broach the topic of Time Series Forecasting as well. Aside from volume, it’s also possible to collect data from different sources. /pol/ is logically not the only board on the website, it’s simply one of the most popular and volatile. It’s possible to research the other communities on 4chan as well. This step can be formulated in the following SMART way:

*“Expand upon the data acquisition process by not only collecting over a broader time period of 1 to 8 weeks, but also by collecting data from the remaining top 4 of the top 5 most popular 4chan boards: /vg/, /v/, /vt/, and /b/. Because different boards tend to discuss different topics, it will also be possible to see which topics tend to attract more hateful behaviour, making data more varied.”*

The HateXplain checkpoint has been utilized to classify textual sequences into 3 categories of speech. The implementation has been used through import of the Hugging Face version of this model. However, the entire model and its code are also available in their GitHub repository. It’s explained on GitHub that the model should also be capable of identifying which specific groups are the target of the classified hate, such as minorities, LGBT people, or religious groups. However, the creators are not fully satisfied with the implementation yet and have thus not added this to the Hugging Face version yet. Research can be performed into whether this is possible with the GitHub version of the model, or an eye can be kept on when this feature becomes available. This step can be formulated in the following SMART way:

*“Perform research into the capability of the HateXplain model to classify the targets of hate and determine whether this feature is already available, or when it becomes available in the future. This research is relevant to uncover new model possibilities to produce even better classification results and hate-related insights.”*

It’s also possible to look at different NLP-related tasks that could be performed to improve upon the Data Science project. Right now, only sequence classification has been utilized, and only on textual data. Posts on any of the boards also contain lots of images. These range from simple, humorous images, to detailed infographics that can reveal a lot more about common rhetoric on a board. Text could possibly be extracted from these images to add to the collection of data. It would also be possible to something like token classification to determine whether rhetoric is talking a lot about specific buildings, locations, politicians, celebrities, etc. This step can be formulated in the following SMART way:

*“Perform research into the efficacy of additional NLP-related tasks that could provide value to the investigation of rhetoric and daily life of a 4chan community. Begin with 2 additional methods such as text recognition and token classification. This research will be able to provide additional data and a better understanding of the topics that arise daily in these communities.”*

Finally, there are many more models out there aside from the HateXplain checkpoint. For this project, it has been chosen due to the availability of a dataset with identical labels that could be used for the fine-tuning process. Naturally, many other models exist. Whereas the absence of a viable labelling process limited the options beforehand, the fine-tuned HateXplain checkpoint could now potentially be used to label your own data on normal speech, offensive speech, and hateful speech. By exploring more models and model types, it may be possible to discover even better alternatives, or models specific to other tasks that deviate from the ones performed so far. This step can be formulated in the following SMART way:

*“Explore at least 5 models to discover potential alternatives and improvements that could be utilized for the achievement of even better results for the understanding of a 4chan community. Because this Data Science project limited itself to a single model, broadening horizons allows for more insight into what the NLP field in its entirety is currently capable of, and for more optimized processes.”*

All these steps have been designed to improve upon the NLP results that have been achieved so far. As the current project serves as a solid foundation, it’s only natural to improve upon it at first instead of branching out to something completely different. Once new results have been achieved, it’s simple to funnel them into the dashboard as well due to its simple click-and-drag functionality. With these steps, I’m confident that overarching business goal will be satisfied even more than it already has been.

**4.4 You review the Data Science process and collect lessons learned on the process & product**

Finally, a review of the overall project can be done to see what worked and what didn’t. It’s good to consider what to avoid, and what could hold potential for the future. It’s important to note that the rubric for this section of the document requires for lessons learned to be based on the sprints that have been performed. However, due to this being a solo project without a team or stakeholders, and due to being busy with an internship at the same time, I have not worked with a schedule of sprints at all. For this reason, the lessons learned are more loosely defined.

The project was an educational experience and proceeded smoothly overall. Due to NLP being a completely new topic, a lot of research into each of the related parts of NLP models was necessary. Luckily, the Hugging Face website offered lots of tutorials, courses, and documentation related to the Transformer models that they advocate for. Since this specific project has been going on while also performing work for an internship, it has been difficult at times to juggle with the time constraints. However, the interesting topic of NLP allowed for a plenty of motivation that eventually spawned the current achievements. In addition, not being stuck to any specific team or schedule allowed for a better management of time that did end up being available. Naturally, several lessons have been learned, which are as follows:

* Freedom can be stifling at times
* NLP is a very broad subject
* Labelling issues nearly halted the entire project

Due to catching up on this subject, I was put in a unique position that offered an immense amount of freedom. No teammates to keep in mind, no Agile rituals to adhere to, not being forced to work in sprints, etc. This was also to my benefit as I’m currently working on an internship as well. However, this freedom did mean that some of the steps had to be handled in a unique way, such as the creation of the fictional organization The Equal Web. Thus, this freedom meant that everything had to be concocted by myself, including all goals, criteria, models, the direction of the project, etc. While this was enjoyable, it also meant having to deal with doubts about whether things will be of sufficient quality or depth. I think this originated from not having performed enough research into all the requirements, causing moments of hindsight and realization that I forgot certain important elements. To ensure that a similar situation is better handled in the future, I would like to create the following two action points:

* Perform more research into all the requirements and necessities to fully bring a project to completion
* Limit own freedom through upholding a work schedule and milestones to track progress

I have also learnt that NLP is an incredibly broad and complex subject. Even the sequence classification that was performed for this project required knowledge of all sorts of intermediary steps to fully understand what is happening. It was possible to skip all of this through using a Transformer pipeline, but this would go against the goal that I had set for myself with this project. For this reason, all the individual steps have been manually performed and explained in this document. Aside from sequence classification, there are many other NLP topics to practice with, and they may complement each other as well. Thus, in the future I would like to acquire mode NLP knowledge on sequence classification, different models, and different NLP-tasks. For this, I would like to create the following action point:

* Perform more research into the field of NLP as a whole to identify NLP-related tasks and projects that would allow to acquire more useful knowledge of the field

Lastly, the issue of needing specifically labelled data to perform the fine-tuning process correctly almost brought the entire project to a stop. This is something that was not realized at first and was only discovered once more research into the models was performed. At that point, data had already been collected from the 4chan API, and the fictional organization and context had already been written. The discovery of a data source with equal labels managed to be a saving grace. Otherwise, it would have been necessary to seek out other models that can satisfy the context and goals that I had set for myself. And if that would not have worked, it would have been necessary to upturn the project and seek out another subject that would have been possible. This issue originated due to a lack of foresight, which is understandable when you consider that NLP has been a completely new topic until now. As a Data Science process, even NLP underwent many of the usual requirements such as the splitting of the dataset into training, validation, and testing sets. However, there were many more unique requirements to consider than originally thought. To better prepare for situations like these, I would like to create the following action point:

* Perform thorough research into a new topic to fully understand the necessary requirements and avoid common pitfalls

Afterword

Overall, the project was a very education al and interesting experience. It is not something that I had expected either. Due to the freedom provided for this project, I decided that I wanted to dive into something new. Thanks to the recent release of GPT-3 and messing around with it with friends, I got the idea of exploring Natural Language Processing. Following the tutorials and courses on Hugging Face went smoothly overall.

The project did run into many hiccups at times, especially related to the fine-tuning process. It was difficult to figure out exactly what to do for the fine-tuning and how to use the trainer properly. In addition, it was uncertain whether the t-davidson dataset was prepared properly and would produce the desired results. Luckily, it seems that I had performed everything right, and that the results massively improved. Other issues arrived w hen pieces of code took exceptionally long to execute or ate up all my system’s memory. Nevertheless, all these experiences served as valuable lessons. Given the subject matter, it was also not possible to provide proof for all the various elements of the portfolio rubric. I have attempted to supplement each section with as much additional material as possible wherever necessary.

Overall, this entire project came to be due to adventurous spirit. I did not want to copy & paste something that I had already done before, just to get it over with. I wanted to take the opportunity to learn something new and enjoy myself. The topic of classifying hate speech was chosen for the same reason: it’s exciting, a little dangerous, and controversial. I also think that I just didn’t feel like working with shoes or fish again.

I hope that the same enthusiasm has managed to contribute to cobbling together a decent portfolio as well.

References

4chan. (n.d.). *GitHub - 4chan/4chan-API: Documentation for 4chan’s read-only JSON API.* GitHub. Retrieved May 14, 2022, from https://github.com/4chan/4chan-API

Data Science Process Alliance. (2022, May 10). *CRISP-DM*. Retrieved May 29, 2022, from https://www.datascience-pm.com/crisp-dm-2/

Davidson, T. (2017). *hate-speech-and-offensive-language/data at master · t-davidson/hate-speech-and-offensive-language*. GitHub. Retrieved May 11, 2022, from https://github.com/t-davidson/hate-speech-and-offensive-language/tree/master/data

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). *Automated Hate Speech Detection and the Problem of Offensive Language*. AAAI. Retrieved May 10, 2022, from https://aaai.org/ocs/index.php/ICWSM/ICWSM17/paper/viewFile/15665/14843

Davis, D. (2017, September 29). *Perspective | I wanted to understand why racists hated me. So I befriended Klansmen.* Washington Post. Retrieved May 29, 2022, from https://www.washingtonpost.com/outlook/i-wanted-to-understand-why-racists-hated-me-so-i-befriended-klansmen/2017/09/29/c2f46cb8-a3af-11e7-b14f-f41773cd5a14\_story.html

Draelos, R. (2019, September 15). *Best Use of Train/Val/Test Splits, with Tips for Medical Data*. Glass Box. Retrieved July 1, 2022, from https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/

Hugging Face. (n.d.). *Natural Language Processing - Hugging Face Course*. Retrieved June 14, 2022, from https://huggingface.co/course/chapter1/2

Mathew, B., Saha, P., Yimam, S. M., Biemann, C., Goyal, P., & Mukherjee, A. (2021). *HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection. Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(17), 14867-14875. Retrieved May 28, 2022, from https://ojs.aaai.org/index.php/AAAI/article/view/17745

Papasavva, A., Zannettou, S., Cristofaro, E. D., Stringhini, G., & Blackburn, J. (2020, January 21). *Raiders of the Lost Kek: 3.5 Years of Augmented 4chan Posts from the Politically Incorrect Board*. arXiv.Org. Retrieved May 15, 2022, from https://arxiv.org/abs/2001.07487

Uszkoreit, J. (2017, August 31). *Transformer: A Novel Neural Network Architecture for Language Understanding*. Google AI Blog. Retrieved July 16, 2022, from https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need*. Arxiv. Retrieved May 20, 2022, from https://arxiv.org/pdf/1706.03762.pdf