UiA University of Agder

Master

Tsetlin Machine for Fake News Detection: Enhancing Accuracy and Reliability

Bjørn Vetle Ledaal

SUPERVISORS
Bimal Bhattarai, Lei Jiao, Xuan Zhang

University of Agder, 2023
Faculty of Engineering and Science
Department of Engineering and Sciences

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja / Nei
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja / Nei
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja / Nei
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja / Nei
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja / Nei
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja / Nei
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Ja / Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja / Nei
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgements

Tsetlin Machine for Fake News Detection: Enhancing Accuracy and Reliability is a master thesis project by Bjørn Vetle Ledaal for the course IKT590 at University of Agder. The project was supervised by Bimal Bhattarai, Lei Jiao, and Xuan Zhang. I would like to thank my supervisors for their advice, supervision and insights throughout the process.

Abstract

This thesis aims to improve the accuracy of fake news detection by using Tsetlin Machines (TMs). TMs are well suited for noisy and complex relations within the provided data, which on initial analysis, overlaps nicely with characteristics found in fake news. We provide a performant and deterministic preprocessor, which is responsible for tokenizing, lemmanzing, and encoding to a representation that the TM understands. We compare our approach with TMs against Neural Network (NN) models over a variety of well-known datasets within the fake news domain. Our findings show from comparable results to significant improvements over state of the art. Additionally, we show how TMs allow for interpretable propositional logic rules. For datasets with 2 classifications, we further convey these rules during inference by applying a color between red and green, which shows the intensity and what direction each word pulls the classification towards.

Keywords: Tsetlin Machine, Fake News Detection, Drop Clause, Interpretable Pattern Recognition, Propositional Logic

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ML	Machine Learning
NN	Neural Network
DL	Deep Learning
GPT	Generative Pre-trained Transformer
LLM	Large Language Model
TM	Tsetlin Machine
TA	Tsetlin Automaton
TMU	Tsetlin Machine Unified
NLP	Natural Language Processing
PRNG	pseudorandom number generator
FSM	Finite State Machine
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network

Chapter 1

Introduction

In this paper we will go over recent advancements within the TM field. The accompanying source code described herein is available online[1].

1.1 Artificial Intelligence

AI has in recent years shown massive growth within several fields, which has garnered the attention of the public at large. While more primitive AI has been around for a few decades, recent advancements have shown how impactful AI can be, or more specifically the Machine Learning (ML) kind. AI has allowed machines to exhibit intelligent behavior and carry out tasks that would otherwise be relegated to humans. Common areas benefiting from AI in various degrees include; chatbots, virtual assistants, text-to-speech, speech-to-text, object recognition, text recognition, fraud detection, stock market predictions, and recommendation engines. While some of these areas are broad, and some narrow, there is a huge swath of fields which also benefit from ML. What all these fields have in common is that ML has been instrumental to various improvements over the years they have been available.

NNs are a big part of ML, which can be simplified as a system modelled after the biological brain. Terminology is however a bit different, neurons become nodes, synapses become edges, each of which are assigned a weight during training according to a chosen reward function. A new concept called layers can also be found in NNs. These layers are named input, hidden, and output. There is also a variant where more hidden layers are used, named Deep Learning (DL). While this may sound overtly simple, both NNs and DL have a wide array of research improving the state of the art year-over-year, adding complexity in each step in the pursuit of additional performance.

Fine tuning a NN is a difficult task, because the underlying task is itself difficult. Making sense of a NN is complex, and if it is not executed correctly the resulting model could end up learning the wrong thing. Why is this a bad thing if it is ultimately correct? Outside of technological issues, there has been ethical issues raised where a AI would disqualify women from being hired because they were women [13]. That is illegal. Cases such as this one highlight the need for interpretability within ML. Ablation studies are used to address such issues, where some data is hidden from the NN, which through experimentation highlights what data contributes to the overall outcome.

With ablation studies the problem is not entirely solved. In Amazon's case blocking out names of applicants was not enough, the resumes still contained indirect links to the applicant being a woman which was picked up by Natural Language Processing (NLP). A direct

[1]Source code https://github.com/vetleledaal/ikt590

example would be based on the school the applicant went to, which in general seems fine, until it penalizes the resume more harshly if it was an all-women school. This shows there is a need for interpretability within ML which shows causation, and not just correlation.

NLP is not only limited to parsing resumes, a big societal issue exist around the spread of fake news. Social media definitely contributes to the current situation by the ease of distribution [42]. This runs into the same issue of not being able to tell if there is some unintended bias within the ML model. That does not make it impossible to make a fake news ML classifier, you just have to accept an unknown amount of bias. This is not always ideal.

OpenAI's ChatGPT is a commercially available service showcasing the latest iteration of the Generative Pre-trained Transformer (GPT) [26]. GPT is a Large Language Model (LLM) that performs NLP tasks in the form of predicting the next token, step by step. ChatGPT is a GPT in combination with system instructions so it can be interacted with like a chatbot. It has been theorized that a LLM such as ChatGPT can be used to interpret classifications provided by other ML models. ChatGPT appears to be capable of performing such tasks, however, there are some concerns of bias, mostly stemming from its closed nature.

A TM is a fundamentally different approach from a NN, while still being within the Deep Learning field. A TM works by applying automata functions to matrices [17]. The Tsetlin Automaton (TA) learns by applying rewards or penalties based on its current state. In a TM a TA represents a simple building block, also known as a clause. Each clause has two actions it can end up in, Action 1 and Action 2. The action decided upon is dependant on the previous rewards or penalties given based on the overall outcome.

While the description of the TM may seem more complex than NNs, the simplicity lies in how the architecture allows for addition and subtraction operations, as opposed to multiplication operations found in NNs. Multiplication operations are notoriously more expensive to execute. While general purpose hardware can perform quite competently on this task, the architecture allows for a TM to be represented by logic gates, which is suited for purpose built hardware.

A feature of this architecture allows one to directly interpret the decision making through a discrete set of conjunctive expressions in propositional logic. These are inherently easy to parse, especially when compared to NNs. This is a direct solution to the otherwise black box nature of NN models. There are however some drawbacks with TMs which makes it unsuitable for some ML tasks. NLP is however one of the tasks the TM especially excels in [37].

1.2 Fake News

While fake news can be classified through a AI models, that is not the only approach that is available. We will explore past, current, and future methods of classifying fake news. We expect research within this field to be more prevalent in recent years, due to the rise of social media and the spread they enable. This directly benefits us in the sense that we may provide improvements within the field in general, while also benefiting from previous research within the domain.

Fake news comes in many forms, and there are adjacent fields we can draw parallels from. From the simplest form, which we will focus on in this paper, is binary classification. Sadly not everything is black and white, which is where multi-class AI models come in. We will put this into consideration when looking at related work.

Unlike more general NLP tasks, we see a general trend of classifying a piece of news as fake or real as insufficient. This comes from the general distrust placed on fake news classifiers in the first place. This requires additional work around providing a chain of logic that gives additional authority to the initial classification.

Let us take a step back and look at an earlier iteration of fake news detection. Before the prevalence of the internet, news were inherently more difficult to distribute. Distribution methods include; word of mouth, newspapers, radio, TV broadcast, among others. The common point between all these is the relatively limited reach and cost associated with distribution. This allowed established news agencies to proliferate news across their audience. This alone is not directly a surefire way to distinguish fake news from real news, rather it put accountability behind news distribution.

Once the internet was made more readily available to the public at large, news started being available through the medium as well as an alternate distribution platform. The reach of the internet was however larger than all other forms of distribution. Coupling this together with the lower cost associated with spreading news, we saw a natural rise in terms of fake news. It was no longer readily apparent which entities were trustworthy, and accountability was less of an important indicator.

Now, in such an environment, we can fall back to the chain of trust established through referrals. If one directory listing links to a news source in a authoritative manner, we can at least consider the linked resource as trustworthy. This is not a scalable approach. Which is where we see fact checking websites establishing themselves as authorities on a large variety of fake and real news.

This works well on a lower scale, but becomes difficult to maintain once the volume of fake news increases. This is what we see with the rise of social media, which enables spread of news in all shapes and sizes, without much authority behind the news pieces. Which is fine for fact checking websites as long as the volume is low, but alas, it was not meant to be.

It was clear another approach was sorely needed to combat fake news in all shapes and sizes. The natural evolution of detection revolve around heuristics based on intuition. These approaches are hard to fine-tune, hard to generalize, and easy to sidestep once the adversary is aware of the detection method. In later times this has been further extended to the use of ML models. This comes from a combination of available research and more powerful hardware.

It is clear that the spread of fake news will in broad strokes always be prevalent. This only highlights the importance of more efficient fake news detection tools. For the detection tools to be effective, it is imperative that the results themselves are explained. Regardless of what tool is used, manual fact checkers, heuristics, or ML models. All of these classifications hinges on showing a rational reasoning behind them.

We will explore the next step, namely using TMs for the classification and explanation tasks. Since TMs show promising results for NLP tasks, and explainability is a large concern when it comes to fake news detection, we find it worthwhile to look into.

1.3 Goal

The goal of this thesis is to improve speed and accuracy of fake news detection, in addition to providing a framework to more easily work with NLP related tasks. We also showcase the interpretability of TMs, especially in combination with the max-literals variant of TM. We explore how to make a reasonable representation of how the TM executes on its inference.

Our framework would be responsible for preprocessing, training, postprocessing, and hyperparameter tuning.

1.4 Problem Statement

1.4.1 Research Questions

Here we specify what areas this paper explores. These questions will be used as a guiding principle throughout the paper, and are crafted to encourage exploration in the research process. Aligning with these questions allows us to clarify how the smaller details connect to the bigger picture.

- Are Tsetlin Machines more accurate at detecting fake news than other machine learning models?
- Does the Drop-clause Tsetlin Machine variant provide improvments within the fake news detection domain?
- Does the Clause Size Constrained Tsetlin Machine variant provide improvments within the fake news detection domain?

1.4.2 Hypotheses

In this subsection we list out our hypotheses, which we will reflect upon throughout this paper.

- A Tsetlin Machine can outperform traditional machine learning models on the same fake news dataset.
- With Tsetlin Machine's interpretability, it is possible to present how it classifies fake news in a meaningful way.

1.5 Assumptions and Limitations

1.5.1 Assumptions

- We have enough computational power to optimize our models for each dataset.

1.5.2 Limitations

- When comparing performance against other ML models we do not have the luxury of running the same preprocessing steps.
- Interpretability may be difficult to present accurately.
- The binary bag-of-words representation is not very memory efficient, which limit the dataset sizes we can use.

1.6 Report Outline

This section gives a brief overview of the structure of this paper. A brief summary accompanies each chapter.

- **Chapter 2**: **Related Work** around TA and Fake News detection.

- **Chapter 3**: **System** design, how everything is arranged.
- **Chapter 4**: **Results** and how these can be understood.
- **Chapter 5**: **Discussion** about what did not work properly.
- **Chapter 6**: **Conclusion**, overall, limitations, and future work.

Chapter 2

Related Work

2.1 Tsetlin Machine

2.1.1 Standard Tsetlin Machine

A TM mainly consists of TAs, a methodology invented by Ole-Kristoffer Granmo on how to implement Michael Lvovitch Tsetlin's description on how to apply automata functions to matrices [17]. The TA learns by applying rewards and penalties based on its current environment. In a TM, each TA can be considered a building block. In the TM these are also known as clauses, these can be seen in Figure 2.3, which we will go into more detail later. Each clause has two actions it can perform, Action 1 and Action 2. In Figure 2.1, we can see the two actions, the output given from a clause is the action itself and this is derived from the accumulation of rewards and penalties gathered. The overall outcome does not flip-flop based on a single reward or penalty, but rather an abundance of rewards or penalties will be able to switch the state of each clause.

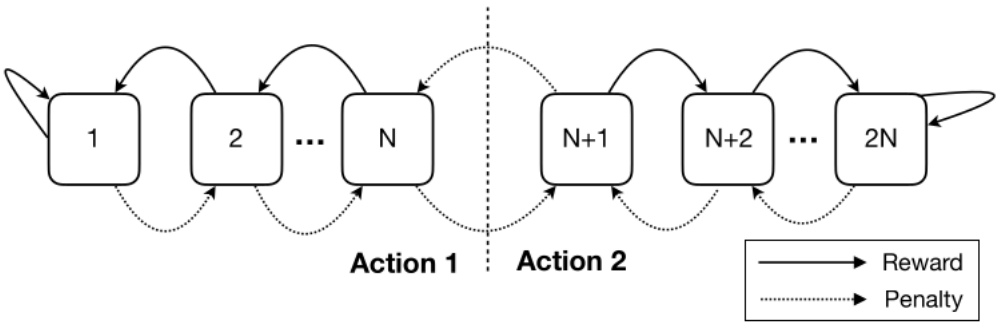

Figure 2.1: "A Tsetlin Automaton for two-action environments" [17].

Figure 2.2 gives an overview for the inference system present in the TM [17]. The input vector consisting of propositional variables is evaluated through multiple conjunctive clauses. The clause size is user-defined, these arer further split into two types of clauses. Clauses with positive polarity and clauses with negative polarity. The figure denotes this with either a "+" or "-" sign. These clauses are used to find sub-patterns in the input data. We also allow clauses without literals, these clauses output 1 during the learning process and 0 during classification. This approach allows these clauses to be updated during the learning phrase, while being deactivated during inference. The majority vote, which gives the output in the figure, is decided upon by having the positive clauses vote for 1 and the negative clauses vote for 0.

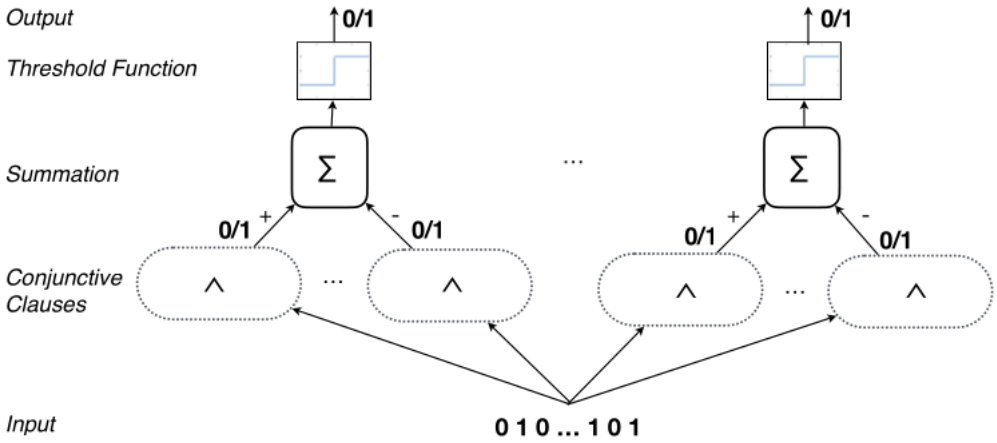

Figure 2.2: "The Tsetlin Machine inference structure, introducing clause polarity, a summation operator collecting "votes"[sic], and a threshold function arbitrating the final output." [17].

We can see a lower level representation of a set of positive and negative clauses in Figure 2.3 [17]. These two clauses have their own set of TA, where each TA has the two actions as mentioned previously in Figure 2.1. Here we name those two actions "Include" and "Exclude", which is decided upon by the state of each TA. Here, each TA has six states. Both of these clauses have four literals, x_1, x_2, $\overline{x}_1$, and $\overline{x}_2$. In the positive clause we note that x_1 is in state 4, x_2 is in state 3, $\overline{x}_1$ is in state 2, and $\overline{x}_2$ is in state 4. Since we have a total of six states, the lower half is excluded and the other half is included. With this, we end up with the effective literals being $x_1\overline{x}_2$. Similarly, for the negative clause, we follow the same process, x_1 is in state 5, x_2 is in state 4, $\overline{x}_1$ is in state 2, and $\overline{x}_2$ is in state 3. From this we see that the TAs that fall into the include category result in x_1x_2.

More generally, we see that Learning Automata has been shown to produce the optimal action in unknown stochastic environments with noisy data[17]. These environments are however limited to small-scale pattern recognition tasks. Outside of previous Learning Automata designs, the TA provides additional improvements within the field; decentralized control, searching on the line, equi-partitioning, streaming sampling for social activity networks, faulty dichotomous search, learning in deceptive environments, and routing in telecommunication networks.

For a TA you need to keep track of a single integer, this decides which action will be taken. By combining multiple TAs you can solve increasingly complex problems through pattern recognition. The rewards and penalties are applied to the integer as either plus one, or minus one. This design is well suited for hardware implementation, as the design lends itself to logic gates [49, 50]. This also carries over the general compute hardware to a lesser degree. Because of this simplicity, we can consider a TA to be a type of Finite State Machine (FSM), which gives us a decent reference point.

This design does however present some challenges, the most prominent one being how data should be represented when taken as input to the TM model. This is of course not a challenge unique for TMs, but still of note because the TM only accepts binary data as input. For NLP tasks the encoding is relatively simple, each word is either in a piece of text or not [36]. Image classification is also possible, but this relies on setting an appropriate threshold to either set each desired feature to a binary value [18].

While challenges are bound to crop up, we do have some advantages from going with TMs. Other NNs approaches will typically run into issues with overfitting, which we can see by

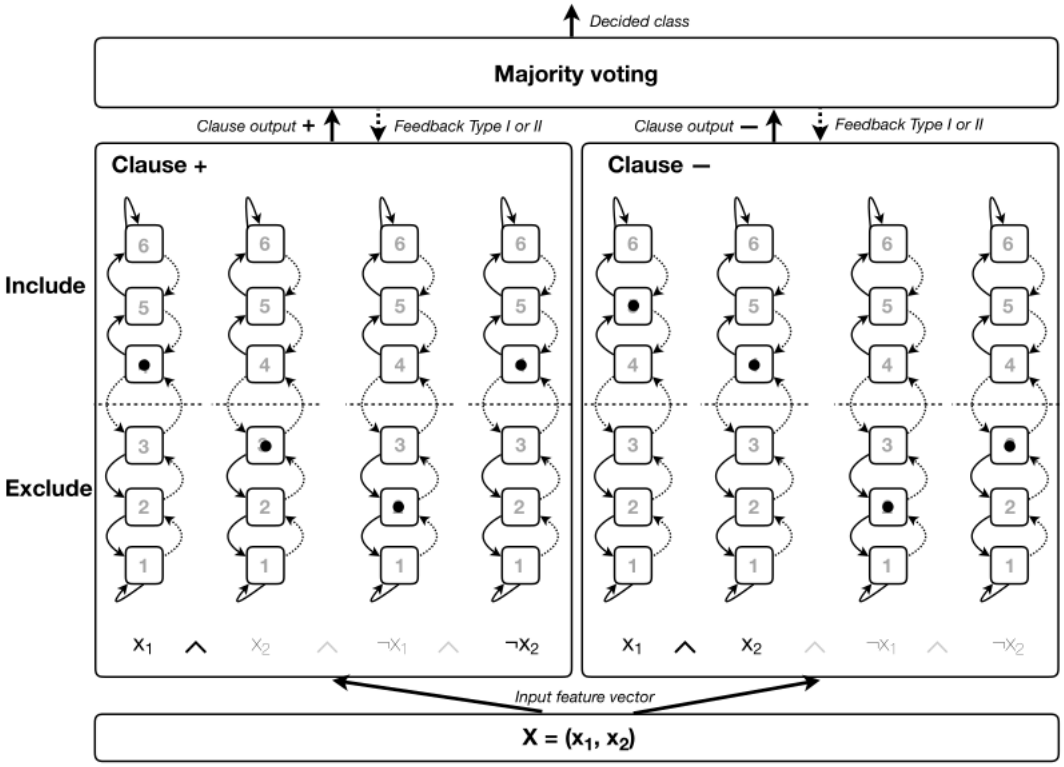

Figure 2.3: "Two Tsetlin Automatons teams, each producing a conjunctive clause. The overall output is based on majority voting" [17].

the model performing extremely well over the training set, while underperforming on the testing set [52]. For the TM we can observe the same suspiciously decent performance over the training set, but still get performance indicative of overfitting. What gives? With high scores over the training set we can instead consider the model "done", rather than actively getting worse [17]. In other words, we can say that the model has extracted everything useful from the training set, and additional training will not yield better results. This does however imply that the model was either provided a simple dataset, or that the hyper-parameters were ill-defined.

Outside of the classification given from the TM, we can peek at the current state of the model. By looking at the clause bank, we have the ability to interpret the reasoning behind a classification [7, 24]. This is a major feature of the TM, and is a clear improvement when compared to other AI models [25]. While there are techniques to get a sense of reasoning from these AI models, the TM is unique in the way it provides reasoning with causation instead of correlation [31, 28, 32]. This has been a major issue for AI, where the model learns the wrong thing [22]. Not only is this important for explainability, but ultimately the overall trustworthiness and reliability is put into question [28].

2.1.2 Drop-clause

In an effort to reduce overfitting and improve the overall performance, as in the AI field, we look to the Dropout technique for inspiration [45]. In a NN, Dropout will at random drop units along with their connections, this is done to reduce co-adaption between units. The drop-clause TM variant proposes implementing a similar technique, where instead of units, it will drop clauses [2]. Same as with Dropout, this is done with a user defined probability. This approach introduces additional stochasticity in the learning phase. While it may not be immediately obvious, we actually increase accuracy while decreasing the training speed.

The increase in accuracy is derived from the remaining clauses being made more robust. For the training speed, the speedup comes from the fact that we overall do less work with fewer clauses.

By the nature of generalizing the clauses the model ends up being more interpretable as a result. This is a highly desirable trait, as it leads to independent logic, which is inherently easier to parse [22]. This is of course more important depending on how the interpretability will be presented to the user, highly dependant on the post-processing steps. Fake news is, among others, one of the domains that massively benefit from interpretability. We can present the literals directly [7], or we can further process it to better show what influences the classification given. Further, reviewing these clauses would allow us to remove unwanted bias or basing its output on some irrelevant, malicious, redundant, or otherwise undesirable hidden patterns.

When compared to ML in NLP tasks, algorithms such as Support Vector Machine, Random Forest, K-Nearest Neighbours and XGBoost shows that the drop-clause variant of TM outperforms them all by a wide margin [40]. When compared to deep learning algorithms such as CNN, LSTM, BERT, and S^2GC it shows the drop-clause model is comparable in accuracy. This is despite the drop-clause model solely relying on bag-of-words encoding, as opposed to the more advanced representations otherwise seen in NNs, such as Word2Vec, Glove, or BERT features.

2.1.3 Clause Size Constrained

TMs natively output as conjunctive expressions in propositional logic. These expressions are directly interpretable. Problems do however occur when the clause bank size is unrestrained, which quickly make the clauses end up being too long to be useful for easy interpretability [2]. Clause Size Constrained TMs implement a soft size limit for the amount of literals that can be used. More specifically once the limit has been reached, "exclude" actions are reinforced. In experiments, this approach reduces the computational complexity, while maintaining comparable accuracy. In empirical experiments, the authors achieves up to +10% increase in accuracy and two to four times faster learning when compared with the standard TM.

The Clause Size Constrained TM variant changes how Type I Feedback is calculated. Type II Feedback is left unchanged. The change requires the clause size to be less than a given constant as a conjunctive condition, and when this condition is not met only the "exclude" is reinforced. As a result, clauses are expelled when it is oversized.

This does not always provide better results overall. If the pattern, or sub-pattern can not be expressed with the amount of clauses given the Clause Size Constrained Tsetlin Machine will fail to learn the patterns in the data. As an example, with a clause limit of 1 it will not learn the XOR pattern as it requires 2 literals, $C_j^i = \neg x_1 \wedge x_2$, which the standard TM is able to learn. With a limit of 2 literals it is however possible for the Clasue Size Constrained Tsetlin Machine to learn XOR. OR is another pattern the Clause Size Constrained TM can learn. Here the minimal amount of literals required is 1, as we can use multiple clauses to represent the sub-pattern, with the help of the hyper-parameter T. The two clauses would be $C = x_1$ and $C = x_2$.

Regarding the soft limit, this may be transiently exceeded. This depends on the training data, the authors mentions how this can occur for the OR operator: "[...] the length of a clause may be over the budget when the system is blocked. Consider an extreme case for OR operator when T clauses have x_1 and $T - 1$ clauses follow x_2. In this situation, due to the randomness, a clause may become $\neg x_1 \wedge x_2$ based on a single training sample. In this

situation, the system is blocked by T and it will not be updated any longer". This is further said to have a low probability of occurring, but it is possible. Another source for the soft limit being exceeded is because of the Type II Feedback. As previously mentioned, Type II Feedback has not been modified to constrain the size of literals.

2.1.4 Coalesced

Normally, a single TM produces a single output. To get multiple outputs at the same time we can string multiple TMs together. This comes with the obvious disadvantage that all the clauses are limited to their own instance of the TM, which creates a lot of duplicate work if there are sub-patterns shared with the outputs. With the coalesced TM, we use Stochastic Searching on the Line along with TAs teams [16]. This allows the clauses to be shared, or in a way merging the different TMs. As for empirical results significantly higher accuracy has been achieved, for example accuracy goes from 71.99% to 89.66% on the Fashion-MNIST dataset with 50 clauses and 22kB of memory usage. This variant also performs well when classes are not balanced.

2.1.5 Convolutional

The major drawbacks of using Convolutional Neural Networks (CNNs) fall into two buckets; high computational cost, and being a black box. The Convolutional TM variant addresses this by instead of employing each clause once, using each clause as a convolutional filter [18]. The Convolutional TM mostly operates on images, as such the images are augmented with coordinate offsets to make it location-aware. With that said, other types of datasets may be encoded as images for this variant. The authors show this is possible with the 2D Noisy XOR Problem, where they encode the data as a 3x3 image.

2.1.6 Regression

The Regression TM addresses the issue of how continuous data can be encoded in a TM, both as input and output [11]. Continuous data is encoded as a set of bits based on thresholds. Empirical results show that the Regression TM is either equal or superior to existing regression techniques over the 5 datasets that were evaluated against. The conversion from continuous input to binary representation is lossless, and the standard TM was modified to sum up the votes from clauses. A modified feedback scheme is also present, to allow for regression. The feedback changes reduces the error margin of the output.

2.1.7 Weighted

This paper introduces the Integer Weighted TM, which aims to reduce the amount of clauses that are required while keeping the same performance [1]. The basic IWTM algorithm works by weakening inaccurate clauses and strengthening accurate clauses. For empirical results, the IWTM uses 6.5 times fewer literals than the standard TM, and 120 times fewer literals than the standard TM when using real-valued weights. This variant also outperforms various algorithms, such as Decision Trees, SVM, KNN, Random Forest, XGBoost, EBM, both in terms of memory usage, but also F1-score.

2.1.8 Relational

Understanding the inherent structure behind data is what the Relational TM builds upon [38]. This brings beneficial results in empirical tests, showing an increase in accuracy from 94.83% to 99.48%. This is because it learns relational patterns in the form of horn clauses. The Relational TM also shows a ten times saving the size of large-scale strucutred knowledge bases.

2.2 Fake News

Fake news are news that are untrue or misleading in the given context[3]. There are a variety of reasons why these exists and what they intend to do. Fake news is commonly stem from; politics, economic gain, psyops, pranks, or satire. Humans shockingly bad at discerning fake news from real news, in a meta-analysis of more than 200 experiments it was found that humans were 4% better at detecting fake news over chance [8]. This shows a clear need for better detection methods, which we have seen an up-tick around fake news in relation to the 2016 US election [9], where 25% of the news shared on twitter the preceding 5 months before election day were deemed to be fake news.

Fake news is not a new concept [19], in essence, fake news encompasses any kind of misinformation, disinformation, or the spread of otherwise false information regardless of the purpose. One of the areas you would otherwise not associate with fake news is probably around pranks or satire. While the intent behind these are clear, it is nevertheless something to be worried about. The infamous "Apple Wave" feature, where you could microwave your iPhone to charge is a pretty infamous example [14]. It got to the point that a spokesperson from Apple had to issue a statement, urging people to not microwave their iPhones. As for misleading, you have the video titled "Ending Women's Suffrage" where the adversary gets people to sign a petition on ending Women's suffrage. This is intentionally misleading as the adversary does not sufficiently explain what suffrage is. Instead preying on the fact that the victim assumes it means the same as the word "suffering".

There are of course more sinister applications of fake news, most prominent ones being around politics. These fake news are used to spread fear or further diverge opinions between people. Depending on the person, they may be more susceptible to fake news than others [8], and further the goals of the one manufacturing the fake news in the first place. For the sake of the TM we ultimately do not need to care about the underlying type of text for classification.

2.2.1 Apparent Indicators

There is a variety of indicators if news are fake or not. Some indicators may include metadata; title, date, image retweets, comments. As for other indicators we can look at how the news are spread, timing of events, grammar, sentiment, emotionally charged, what's the origin, and is the actual contents true or false. We see a lot of these indicators depend on mostly easily manipulable indicators which are easy to spot. There has been some work removing these indicators and for both rule and neural network detection methods accuracy was significantly reduced [20]. This is definitely not easy to necessarily protect against depending on the approach. There has been some promising results when the fake news detection algorithm is aware and mitigates this kind of adversary [20]. Some indicators are multifaceted and need to be aligned for it to indicate anything meaningful, or a hidden pattern if you will. There are a lot of minute details that goes into each indicator, let us consider a few readily available indicators that we may wish to use for detection.

A lot of sites have user generated content, with these sites we can consider the profile of the user posting. Fake profiles have some low-hanging indicators such as; join date, activity, profile picture, followers, follows. If these are available they could be good indicators unless the adversary deliberately went out of their way to mask this to avoid detection. There are harder to detect and are more prone to false positives, but one possible indicator, is gap in activity and sudden activity along with similar looking profiles. This may indicate the profile was either sold or stolen. While some fake profiles are easy to distinguish manually, when automating detection we should instead consider stronger indicators. These are inherently easier to follow.

Another strong indicator would be the domain, especially when the content is mostly originating from the editorial staff and not user generated content. While the publishing platform may be a weak indicator, we should not exclusively look at this for classifying the content. The moderation policies also plays a role in how this is evaluated. Some platforms have different federations, so the indicator has to be subdivided into multiple pieces. Segments may be; front page, opinion piece, user-generated news, forum subpage. These should be treated differently as they convey a different kind of approval from the overall website. The specific author and profile can also be analyzed, if available. A weak indicator, the date, can also be considered. This is a multifaceted indicator which does not mean much by itself. As an example, the news article "Queen Elizabeth II has died" was true as of 10 September 2022, but the same can not be said if the date was 10 August 2022. There is of course some overlap where there is some ambiguity, which can be better corroborated with other stronger indicators [41]. An excessive spread of news could mean it is either very prominent, or appearing to be so. This indicator is somewhat ambiguous and is hard to explain unless the evidence disproving it is very strong.

2.2.2 Detection with Neural Networks

There are multiple nuances to detection fake news, as such there are multiple sites providing fact-checking services. The most prevalent and purpose-built ones include Snopes, PolitiFact, Poynter, and GossipCop. These are not the only sites contributing to fact-checking of news, some news sites have a section dedicated to fact-checking. For example, Entertainment Weekly, People Magazine, RadarOnline. The major downside to this kind of fake news detection is the ambiguity around having to produce an answer, when often it is more complicated than that [48]. Even when interpretable results are present, it often paints a detrimentally simplistic view of whatever that is being fact-checked. The authority of these sites hinges mostly on the accuracy of their reporting [15].

There is still some value in these sites, they provide a source-of-truth from a variety of experts within their domain. It is both useful for end-users, and for training ML models. although present in all fake news detection methodologies, bias is a concern for fact checking websites. Out of 267 statements fact-checked by Washington Post, made by presidental contenders in the 2012 running, only 7 were deemed fully true [48]. That is 3%, which is shockingly small, to the point that the evaluation is put into question. These fact-checking sites do in any case produce useful data for use withing NNs, which is more and more important with the ever expanding rise in user generated content [33, 10].

While building on top of manual fact checking gives a high accuracy score, this is not always the most practical approach. Rule based fake news detection relies on heuristics set up by the tool author. These may work very well, but requires constant fine tuning as adversaries adjust their behavior to avoid detection [20]. Manually updating these rules are a cat-and-mouse game, which lasts as long until one party gives up. These rule based approaches do however provide the advantage interpretability, since they are naturally built on top of logical rules. This can be considered a stop-gap solution until something better comes along, but while fake news detection is lacking, this is a decent compromise given the limitations are known.

Outside of manually maintaining either a large corpus of fact-checking articles, or an increasingly complex set of rules, we can use the prior art found in these approaches and apply them to ML. The detection models derived from the available datasets is in any case an useful resource for end-users, furthering ML algorithms, and journalists. For journalists specifically they use these models for rapid fake news detection, in conjunction with traditional journalistic research [10]. Not all NNs are suited for processing these datasets, the main concern here is the limited size of the dataset. While it is possible for a NN to learn with

few-shot, or one-shot learning, this is not prevalent through every ML model [47]. The norm is rather that NNs require a lot of data to produce useful results. It is not uncommon that fake news datasets are limited to the tune of less than 10.000 datapoints [43, 39, 34, 4, 46]. In some cases the NN models otherwise weak averse to few-shot or one-shot learning may be able to overcome some of these limitations with some data augmentation [47]. Some also experiment with transfer learning, where you have a pretrained model that was trained on a related task, and transfer that knowledge to the fake news domain [35]. The type of NN used is somewhat important to the final accuracy results, in the sense that with a decent dataset models would perform within the same ballpark, while still distinguishing themselves enough so we can say they are outside the margin of error. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown results for fake news detection.

With regards to understanding the reasoning behind a classification, we need to compare our results to In a user study [21] the authors compare the accuracy between humans and a NLP classifier. The underlying NLP classifier is not important, but of note the model was more accurate than humans when tested in isolation. When humans had access to the model's classification and highlighted weight on words humans were in general overconfident in their own ability to discern fake reviews from real reviews. The model itself ultimately far outperformed the human counterpart. While reviews and news are not exactly the same, both are NLP tasks which are reasonably similar to draw the same conclusion. This does however tell us more work needs to be done on explainability and lift the credibility of the models for these to be effective tools.

2.2.3 Detection with TM

While the TM is based on propositional logic (rule-based) [17], it does not require manually set up of the heuristics, instead this task is left to the training stage where a set of rules are made. This approach has the advantage of being interpretable [5], the same feature gathered from rule-based approaches without the additional manual labor. TMs has a very low inference time [51], and have the ability to work with noisy and generally incomplete data [17], which is very prevalent for all types of news.

Fake news detection is however a largely unexplored area in connection with TMs. However, with the research that is available, we see a significant 5% increase in accuracy when compared to previously published baselines [6]. When compared against BERT and XL-Net the TM implementation does however achieve a slightly lower accuracy score. Despite these results, for TMs we can lean on interpretability, which is a decent contribution within the fake news detection field.

Of importance in the prior art, we must take special care to not misconstrue what the interpretable literals convey. An example that is brought up is the literal "¬trump", this is not an indicator that all content with this literal is fake news. This is instead a useful enough descriptor/discriminator for the model to make its classification.

2.3 Datasets

There is an abundance of fake news datasets, so we can be picky in what exactly we chose. Our goals include; using a well-know dataset for easier comparison, a variety of sizes, a variety of sources, a variety of formats, English datasets. These criteria allow us to evaluate the TM itself, and not the quality of the dataset. We look at the performance trends to evaluate if we met these goals.

For all these datasets, we will define what each entity is responsible for as to not add to the confusion. The article is the actual article being classified. The fact-checking article

classifies the article. In some cases we also have the referenced fact-checking article, which is used when the fact-checking article outsources their classification to a third party. We will expand these definitions with adjectives as appropriate, while making what entity we are talking about clear.

2.3.1 FakeNewsNet

The FakeNewsNet dataset is provided as a collection of Comma Separated Values (CSV) files [43]. This dataset is based on scraped data gathered from PolitiFact and GossipCop. PolitiFact links directly to the relevant news article, or in the case of GossipCop the articles are correlated from a web search as the source is not directly provided. The authors also observed that 90% of the GossipCop ground truths were scored 5 or below, so additional news were scraped and added to the dataset to balance it out. The authors provide a general overview of the dataset, as seen in Section 2.3.1.

The data is provided in the same format for both PolitiFact and GossipCop; `id`, `news_url`, `title`, and `tweet_ids`. The `id` attribute consists of a prefix followed by multiple numerals. The prefix is either "politifact" or "gossipcop-", the number appears to have no direct relation to its origin. The `news_url` attribute contains a URL linking directly to the target article. The `title` attribute is the headline as presented from the target article. Lastly, the `tweet_ids` contains a space separated list of tweet IDs that link to the target article.

We take note of a general trend of separating the dataset into PolitiFact and GossipCop and gathering metrics in isolation [43, 30, 44], with the minority of research combining these [12]. We see this as a positive by the fact that there are more points to compare against.

	Category	Features	PolitiFact		GossipCop	
			Fake	Real	Fake	Real
News Content	*Linguistic*	# News articles	432	624	5,323	16,817
		# News articles with text	420	528	4,947	16,694
	Visual	# News articles with images	336	447	1,650	16,767
Social Context	*User*	# Users posting tweets	95,553	249,887	265,155	80,137
		# Users involved in likes	113,473	401,363	348,852	145,078
		# Users involved in retweets	106,195	346,459	239,483	118,894
		# Users involved in replies	40,585	18,6675[sic]	106,325	50,799
	Post	# Tweets posting news	164,892	399,237	519,581	876,967
	Response	# Tweets with replies	11,975	41,852	39,717	11,912
		# Tweets with likes	31692[sic]	93,839	96,906	41,889
		# Tweets with retweets	23,489	67,035	56,552	24,995
	Network	# Followers	405,509,460	1,012,218,640	630,231,413	293,001,487
		# Followees	449,463,557	1,071,492,603	619,207,586	308,428,225
		Average # followers	1299.98	982.67	1020.99	933.64
		Average # followees	1440.89	1040.21	1003.14	982.80
Spatio-temporal Information	*Spatial*	# User profiles with locations	217,379	719,331	429,547	220,264
		# Tweets with locations	3,337	12,692	12,286	2,451
	Temporal	# Timestamps for news pieces	296	167	3,558	9,119
		# Timestamps for response	171,301	669,641	381,600	200,531

Table 2.1: Overview of data available from the FakeNewsNet dataset [43].

2.3.2 FakeCovid

The FakeCovid dataset is provided as a single CSV file [39]. The dataset is a collection of fact-checked articles and multiple related attributes as seen in Table 2.3. These articles all relate to the SARS-CoV-2 virus, and are gathered from Poynter and Snopes. The article contents were scraped directly from the article as linked from the referenced fact-checking article. Article titles are gathered from the article, along with the article contents. The publishing date, class, and related social media posts are gathered from the fact-checking article. The social media posts are further augmented by finding related posts, such as by

using keywords and regular expressions for filtering. A general overview of the datase can be found in Table 2.2a. Attributes and their use case can be seen in Table 2.3, and the inter coder reliability of the `category` attribute can be seen in Table 2.2b.

Element	Count
Fact checked article	5182
class of fact check article	23
Fact check Category	11
Fact checking website	92
Country	105
Language	40

(a) Dataset details.

Annotator	English	Hindi	German
1st	2116	–	47
2nd	–	114	23
3rd	100	46	–
ICR	91%	96%	94%

(b) Inter-coder reliability.

Table 2.2: Overview of the FakeCovid dataset [39].

Attribute	Description
`ID`	Incrementing ID with no connection to the article.
`ref_category_title`	Title from fact-checking article with class prefix, if known.
`ref_url`	URL of the fact-checking article.
`pageid`	URL of the page linking to the fact-checking article, if known.
`verifiedby`	Name of the referenced fact-checking article.
`country`	Country of the referenced fact-checking article, if known.
`class`	Classification by the fact-checking article.
`title`	Title from fact-checking article.
`published_date`	Published data from fact-checking article.
`country1`	Same as country, if known.
`country2`	Country 2, if known.
`country3`	Country 3, if known.
`country4`	Country 4, if known.
`article_source`	Referenced fact-checking article.
`ref_source`	Name of the fact-checking site.
`source_title`	Title from referenced fact-checking article.
`content_text`	Contents from referenced fact-checking article.
`category`	Annotated category, if known.
`lang`	Language of the referenced fact-checking article, if known.

Table 2.3: Attributes used in the FakeCovid dataset [39].

2.3.3 HateXPlain

The HateXPlain dataset is provided as a JavaScript Object Notation (JSON) file [27]. The data originates from the social networks Twitter and Gab, while the annotation is provided via Amazon Mechanical Turk, which is a service that provides data entry at scale.

Since this dataset is formatted as a JSON file, it can and is stored in a more complex hierarchy than would otherwise be possible with CSV. The JSON file contains one object, this object is not entirely semantically correct JSON, which could result in issues, depending on the parsing library. Each key in the root object consists of a numerical ID with the postfix "_twitter" or "_gab". This object contains; `post_id`, `annotators`, `rationales`, and `post_tokens`. We will go over these one by one.

`post_id` is the same as the key of this object. The `annotators` is a list, the objects contains the key `label`, `annotator_id`, and `target`. `label` is a string that can be used as a classification. `annotator_id` is the numerical id of the annotator providing the label. Finally

`target` is a list of targets the label is directed at. For example, the label can be "hatespeech", "offensive", "normal", etc. The target can be "Asian", "Caucasian", "Women", etc.

The `rationales` and `post_tokens` lists are tighly connected. `rationales` consists of one list per annotator, of which consist of integers that are either 1 or 0 depending on if the annotator finds the respective token from `post_tokens` in the post. The size of each annotators rationales is the same as the `post_tokens` list, which is a list of tokens in the form of strings.

2.3.4 Other Datasets

The **FakeNewsAMT** dataset is provided as multiple text files, one file per post [34]. The class is derived from the file path. The news are categorized into the domains: "technology, education, business, sports, politics, and entertainment. The file names indicate the news domain: business (biz), education (edu), entertainment (entmt), politics (polit), sports (sports) and technology (tech)" [34]. These domains are encoded into the filename alongside the classification. Each file consist of the article title on line 1, and the article contents from line 3 until the end of the file.

The news were scraped from well-known news sites, mainly within the US. The sites include, but not limited to: "ABCNews, CNN, USAToday, NewYorkTimes, FoxNews, Bloomberg, and CNET" [34]. Classification of news was done by assuming the news only contained real news, while the fake news were derived from those. The fake news were produced with manual labor though Amazon Mechanical Turk.

Class	Entries	Av. Words/Sentences	Total Words
Fake	240	132/5	31,990
Legitimate	240	139/5	33,378

Table 2.4: Class and word distribution for the FakeNewsAMT dataset [34].

The **Celebrity** dataset is provided as multiple text files, one per post [34]. The class is derived from the file path. These news are scraped from mainstream news sites, but limited to the entertainment industry. Additional real news were gathered from entertainment magazine websites. Fake news were gathered from "gossip websites such as Entertainment Weekly, People Magazine, RadarOnline, and other tabloid and entertainment-oriented publications" [34].

After roughly dividing news into two classes, the authors manually verified the classification through fact-checking websites such as GossipCop, and cross-referenced against other news sources. The different

Class	Entries	Av. Words/Sentences	Total Words
Fake	250	399/17	39,440
Legitimate	250	700/33	70,975

Table 2.5: Class and word distribution for the Celebrity dataset [34].

The **Election Day** dataset is provided as a single XSLX file [4]. This dataset can be used as both for binary classification and multiclass classification. This is a collection of viral tweets gathered on 8 November 2016, the US election day. Viral tweets are defined as any tweet receiving 1,000 or more retweets. The tweets were gathered from Twitter by using the search terms #MyVote2016, #ElectionDay, #electionnight, @realDonaldTrump and @HillaryClinton. A lot of attributes are provided, for a variety of use cases, such as `is_fake_news`, `fake_news_category`, `tweet_id`, `created_at`, `retweet_count`, `text`, `user_screen_name`,

`user_verified`, `user_friends_count`, `user_followers_count`, `user_favourites_count`, `tweet_source`, `geo_coordinates`, `num_hashtags`, `num_mentions`, `num_urls`, and `num_media`.

The **FakeNewsChallenge** dataset is provided as multiple CSV files[1]. This dataset provides multiple classes, and consists of 49,972 records. Fields used for this dataset is limited to `headline`, `body`, and `stance`.

The **FakeNewsCorpus** dataset is provided as a single CSV file [46]. In this dataset, we are provided the fields `id`, `domain`, `type`, `url`, `content`, `scraped_at`, `inserted_at`, `updated_at`, `title`, `authors`, `keywords`, `meta_keywords`, `meta_description`, `tags`, `summary`, `source`. The data is sourced from the website OpenSources, which describes itself as a "Professionally curated lists of online sources, available free for public use." [29]. The authors of the FakeNewsCorpus dataset used the classification gathered from this website, along with scraping the linked articles over the 1001 domains present. Because of the unbalanced dataset, namely a lot of fake news, the dataset was augmented with news from NYTimes and WebHose English News Articles. The dataset is still a work in progress, so the datasets are versioned. The version described herein is version 1.0 of the public dataset. This dataset is limited to 9,408,908 articles (745 out of 1001 domains).

[1] From http://www.fakenewschallenge.org/

Chapter 3

System design and configuration

Unit	CPU	GPU	RAM
Local	Intel Core i7-5960X (16) @ 3.00GHz	GeForce GTX 980	32 GiB
UiA 1	Intel Xeon Platinum 8168 CPU (96) @ 2.70GHz	Tesla V100-SXM3-32GB	1.5 TiB
UiA 2	Intel Xeon Platinum 8168 CPU (96) @ 2.70GHz	Tesla A100	1.5 TiB

Table 3.1: Hardware configuration details.

3.1 Overview

All training for this paper has been carried out in the systems outlined in table 3.1. The remote systems are provided by UiA, are assigned at random, and has been the main systems for training the ML models. Due to some workflow issues with the remote systems, the local system provided by the authors has been used for parts of training.

We can see the general workflow in fig. 3.1. We can pick between starting with `matrix_search.py` or `main.py`. `matrix_search.py` consist of repeatedly running `main.py` in the search for the optimal hyper-parameters given a dataset. If we decide to run a single set of hyper-parameters, we start with running `main.py`. This script is split into 2 stages, preprocessing and training. Preprocessing includes creating a vocabulary for the given dataset and filtering out undesirable words. Training consists of advancing the TM step-by-step one epoch at a time, while calculating the performance.

In the postprocessing stage we are trying to understand the data produced in the previous stages. This is an interactive workflow, and used to draw conclusions and comparing against other models. In `plot.py` we produce graphs in the form of interactive graphs and tables we can directly embed in LaTeX. These can be seen in the results section. In `explain.py` we extract the state of the previously trained model, together with the accompanying vocabulary. With this we can evaluate the interpretability of the model. To properly present how the model classifies, we take proper scaling and color choices into consideration. The complete text is either shown through a browser with HTML, or PDF, depending on the target use case.

3.2 Preprocessing

Preprocessing consists of making the datset consistent regardless of source, creating a vocabulary given the dataset, filtering out words that were not used at least 3 times, and discard the least common words until the vocabulary reaches the vocabulary limit. Sorting for the most

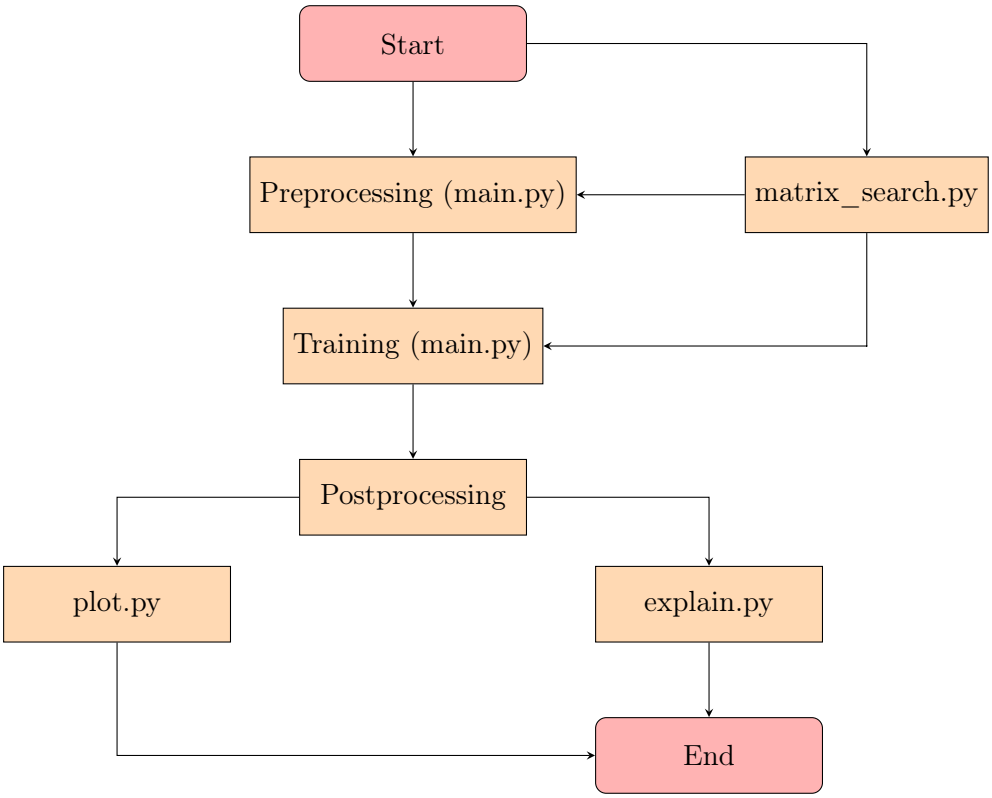

Figure 3.1: Process flow diagram.

common words is deterministic as it uses a set pseudorandom number generator (PRNG) seed.

To make the dataset consistent we have defined a mapping for for each dataset we support. Each dataset has their own method of being parsed, but in broad strokes we load the dataset into a pandas DataFrame, which initially has their own unique schema, which we standardize for code reuse purposes. Not all datasets are straight-forward to parse, such as with the HateXPlain dataset, which is read from a JSON file. Minor semantic differences are ignored by the necessity of loading different datasets.

For all the datasets other than HateXPlain it has been relatively more simple. We get a flat structure, in either CSV or Excel format. The pandas library reads both of these filetypes, which is what we use to load the respective files into a DataFrame. For some slightly more complex datasets we have multiple files, in which case we combine the DataFrame's together. For some datasets the class is given in the file path, while some have these directly available in the file contents. We account for all these cases, as instead of using a generic parser we use purpose built parsers for each dataset. This is done to cut down on complexity.

Since the HateXPlain dataset is not straight forward to parse, we will go over this in a bit more detail. The JSON file consists of an array containing objects that represent one post each. The contents of each post is represented as a list of tokens. For consistency with other datasets we undo these tokens so it can later be tokenized under the same rules. There can be multiple labels affixed to each post. There can be multiple labels, one per annotator. Inside of each label it signifies the target group, which can be for example "Asian", "Women", "Caucasian", etc. If there are no labels set, we set the class to "None". If there are multiple targets, we count how many unique targets are present. If we have more than one unique target we set the class to "Multi". Otherwise, we set the class to the most common label, of which there should only be one. This design allows us to more easily make changes in the future, due to the variety of interpretations here. Since there are multiple classes for

the HateXPlain dataset, we have a flag to 2 classes, or a boolean if you will. We use this for explainability. We are aware that throw away a lot of context by doing this, but a goal of loading HateXPlain was not to make a good hate-speech classifier, but rather make it comparable to the other datasets.

We limit ourselves to creating the vocabulary over the training set only, the test set is not looked at all in this stage. This is done to prevent "cheating", as otherwise the accuracy score would be artificially boosted when in the real world we do not have access to this data. We took data augmentation with external data into consideration, but ultimately decided against it to make the results more plainly comparable for other projects.

When tokenizing the dataset for the vocabulary, we start with removing English stop words as defined in the Natural Language Toolkit (NLTK) library. Words are split by the `str.split()` function[1]. Each word is sent through the same process. While simple to implement, this is also simple to compare results directly. Tokeninzing using n-gram was considered, but ultimately decided against due to concerns of collision on the binary bag-of-words representation.

1. Lowercase the word
2. If found in cache: return cached processed word
3. Remove punctuation as defined in Python's implementation of "string.punctuation"[2], except for the single quote character
4. Strip all consecutive single quote characters at the start and end of the word
5. Lemmatize the word with NLTK's WordNetLemmatizer
6. Discard the word if the length of the word is less than 3
7. Discard the word if the word entirely consists of number characters
8. Save the word to the cache
9. Return the modified word

We encode the train and test sets into a format the TM can understand. We considered splitting into a validation set as well, but decided against it as some of the datasets are comically small. For textual content we represent this as a binary bag-of-words matrix. This also applies to other features such as the domain name, and tweet IDs as the underlying structure is the same, regardless of the misnomer bag-of-words. It would be more accurate to name it bag-of-features, but this is not widely used within ML. The binary bag-of-words is represented as a dense matrix, where each feature is initially set to 0, but set to 1 if that feature is present in the input. It should be noted that them major downside of this approach is that word order is lost.

Finally we encode the classification using `sklearn`'s LabelEncoder. This is mostly a convenience function that works for any amount of classes. We take care to not fit over the test dataset, and instead just encode this part so no new knowledge is acquired from new classes present in the test set, but not the train set. This is somewhat more important when we do not have a validation set, which is the case.

[1] Default behavior is defined as "[...] runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace"

[2] Defined as "ASCII characters which are considered punctuation characters in the C locale"

3.3 Training

Before training we go through some sanity checks, such as making sure the input is in the proper shape. the Tsetlin Machine Unified (TMU) library does not properly catch such issues, and it's behavior in these cases results in undefined behavior. While this is not strictly necessary, it has been useful during development of the preprocessing stage. Additionally we check if there exist a previously fully trained model as to not duplicate work when it is not necessary.

When training a TM we do have a lot of overlapping terminology with NNs. The model is "fit" over the training set, once per epoch. While fitting, we measure metrics that is useful for comparing our model against others. In order, we perform the steps; fit over training set, predict over training set, predict over testing set. We note down the time spent on each of these steps. Fitting is essential, but predicting over the sets is purely for calculating the model's performance. At this point we have the predicted output in an array, we use this for comparing against the ground truth. We use the sklearn library to calculate the metrics; accuracy, precision, recall, and F1 score. These metrics along with time spent is output in standard output of the program. This is either parsed directly, as it is human readable, or through the matrix search script the standard output is saved to file for future postprocessing. Regardless, we save the model to file once it has gone through all the 150 epochs.

We have considered skipping metrics, and ultimately found that is was not worth it. The metrics are necessary for both presenting our findings and more easily find the optimal hyperparameters. We can selectively skip inference based on what epoch we are on, but this is not optimal as graphs would have gaps where we do not test for metrics. If it was important enough we could save each state of the TM and calculate the metrics out of order. With the current design each epoch requires approximately 630 MiB of disk space, or 80 MiB when compressed, this is for a relatively small dataset of 500 entities and 4922 tokens as vocabulary.

The TMU library allows us to set up the TM with multiple variants at once. For simplicity sake we will go over how each variant changes the training stage in one way or another. As we know, the drop-clause TM variant drops clauses randomly given a percentage. The drop-clause variant significantly improves the performance [40], both in terms of speed and accuracy. The speed comes from doing less work, and accuracy is improved by the resilience the remaining clauses. This depends on the task at hand, but has shown promising results for NLP tasks in general which is exactly the use case we are working with here. It is in any case important variant to explore, as it seemingly only have positive traits. For experimentation with this, the percentage of clauses to drop is configurable.

Since we put explainability in focus, we need a way to present this in a reasonable manner. The standard TM does not support culling the amount of clauses that should be used. With the max-clause TM variant we can however do exactly that [2]. For this purpose we made this a configurable hyperparameter. Since both the drop-clause and max-clause TM variants are implemented in the TMU library, we simply pass the hyperparameter supplied from the command line to the TMU library. There are little to no other considerations that are required here.

3.4 Postprocessing

In the postprocessing stage we have a fork in the road on how we want to process it. Presenting the model performance in the form of graphs and tables is the most straight forward one,

which we will start with. This allows for evaluating the model across the set hyperparameters, but also against completely different ML models as long as they use the same dataset and a reasonably similar preprocessing stage. We do not put much weight on other models preprocessing stage unless the TM ends up vastly underperforming in terms of accuracy.

To make present the model both as a graph, and as a table, we parse the previously produced log files. The log files are directly human readable, and must be parsed before the information can directly be mangled by Python again. We chose to use a regular expression for this, as the pattern was strictly regular. There are other approaches, but this is the most straight forward one without having to make another log format, that is intended to be read directly by Python.

While making the table and graph are completely different tasks code wise, for simplicity we make these at the same time as to not accidentally compare out of sync data. For the graphs, we chose to make interactive graphs. We found the plotly library had a decent interface for making such graphs. In addition it could export to Scalable Vector Graphics (SVG) files, which are easy enough to integrate into LaTeX, an example can be seen in Figure 3.2. While most of the metrics can be made out, other graphs are not as easy to parse. From the top left dropdown menu you can pick one trace, which will be the only one shown. Not visible in Figure 3.2 is that the dropbox shows the mean for each trace.

While the mean provided in the interfactive graph is an estimate, we have a more complete overview in the table we write out. This table is again made to be easy to integrate into LaTeX. Examples of Figure 3.2 represented as a table can be seen in Table 3.3

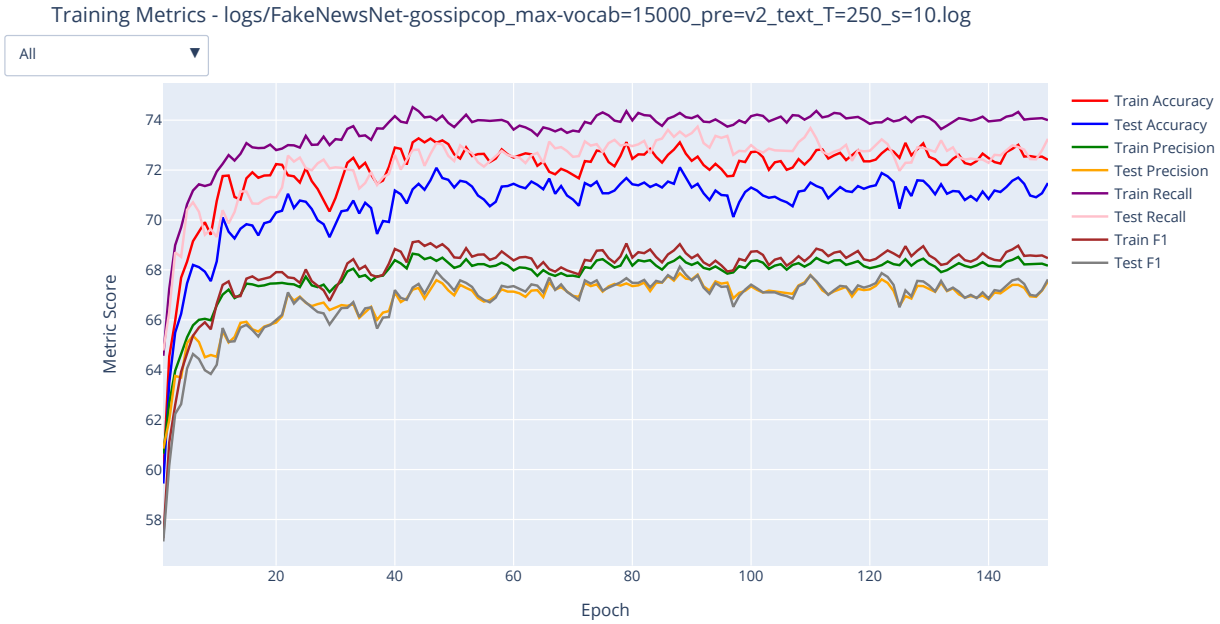

Figure 3.2: Example graph, showing metrics.

3.5 Reproducibility

While presenting usable results is a goal, it is equally as important as being able to achieve the same results at a later date. The hard part about reproducibility is adding it after the fact. Being a concern from day 1, we took this into consideration in each stage, and made sure the same code produced the same output. The only nondeterministic part of the code currently resides in the TMU library. Sadly, the TMU library (v0.8.1) does not support

Table 3.2: Metrics

Metric	Epoch 1	Epoch 150	Δ% last	Max	Δ% max	Std Dev
Train Accuracy	59.80	72.41	21.09	73.27	22.53	01.53
Test Accuracy	59.44	71.48	20.26	72.11	21.32	01.49
Train Precision	60.64	68.17	12.42	68.65	13.21	01.00
Test Precision	60.84	67.53	11.00	67.86	11.54	00.98
Train Recall	64.56	74.00	14.62	74.52	15.43	01.22
Test Recall	64.77	73.25	13.09	73.74	13.85	01.24
Train F1	57.26	68.47	19.58	69.15	20.76	01.40
Test F1	57.12	67.62	18.38	68.13	19.28	01.36

Table 3.3: Example table, showing discrete values.

seeding the PRNG or otherwise making the underlying TM deterministic. With that in mind, we still found it an important feature to consider.

From the beginning we started off with the Poetry package and dependency manager. Poetry allows us to use a virtual environment, where we control all the dependencies to a high degree. With Python's standard pip and `requirements.txt` approach, you have by default very lax version pinning. If you pin the version of direct dependencies in `requirements.txt` you are still not making reproducible builds. The problem lies in the dependencies of your dependencies. These are not pinned, but rather defined by your direct dependencies what version ranges they support. Poetry solves this mess by providing creating a lock file specifying each dependency, their version, and their SHA256 hash. This ensures reproducible builds. When it comes time to update dependencies Poetry has some workflows that makes it easier to work with there as well.

While reproducibility is in most cases a desirable trait, the command line interface allows for setting the PRNG seed if the output produces highly abnormal results, and you need to check if it is a fluke by picking another seed. This is also a pitfall that you may run into, that you optimize your hyperparameters for a specific seed, instead of optimizing it for the typical output. Of note we have a total of two random state consumers. The first one is when splitting the dataset into a training and testing set. The other is when calling `tm.fit()`. The former behaves as expected, the same input produces the same output. The latter does not like to behave, and needs to be accounted for in the TMU library.

3.6 Dataset

A few datasets are handled on how they should be loaded into the software pipeline. The varying datasets are mostly made available over various git repositories, and commands are provided to gather the corresponding datasets. The datasets chosen here provides different methods of loading the data, which is why they are a good fit to show how a variety of datasets can be loaded, independent of the dataset structure.

3.6.1 FakeNewsNet

We use all the attributes we are provided, except for the `id`, which does not provide any useful information.

3.7 Hyper-parameter tuning

All the hyper-parameters can be changed. A script is also provided to do a matrix search over the different hyper-parameters in a search for the optimal set. As always, with larger datasets and a lot of hyper-parameters to check, it may be practically infeasible to find the optimal set of hyper-parameters.

The available parameters include the following;

- num-clauses: TMU param, clauses used
- T: TMU param, threshold
- s: TMU param
- epochs: How many epochs to run for
- device: Either "CPU" or "GPU"
- seed: Seed used for PRNG for reproducability
- dataset: Dataset used for training
- feature: Input to the Tsetlin Machine
- test-size: What fraction of the dataset is used for test metrics
- malformed: how to handle invalid data in the dataset. Either fix or drop
- max-vocab: How many features should be set up for vocabulary
- max-domain: How many features should be set up for domain names
- max-tweet: How many features should be set up for tweet IDs
- preprocessor: What preprocessor to use. Use v2
- drop-p: Fraction of clauses to randomly drop
- max-literals: Soft-limit of literals, used for explanability
- dry: Just show info about the dataset and execution plan

Not all of the available parameters are used for the matrix search, such as epochs, device, seed, test-size, malformed, and preprocessor. These should stay the same to make a reasonable comparison between the chosen parameters. Using these parameters with matrix search results in undefined behavior. As part of hyper-parameter tuning, we run several processes at once to speed up the operation. We use Python's built-in multiprocessing library to achieve the desired result. For each run, we directly execute the CLI interface of the main script. This is to ensure we are using the exact same code path as when invoking the script manually. The alternative would be importing the main script as a library, but this could end up with subtle differences from the differences in how the script is invoked.

When scaling up to training multiple models at once, we expect to see a decrease in the time required. The batch size is configurable to allow for experimentation to find the optimal size given the hardware.

3.8 Explainability

Since TMs are natively interpretable, which we can use to explain the reasoning behind any classification, it would be amiss to disregard this feature, which is more-or-less provided for free. When we use the weighted TM variant, which we do, we use the weight to measure the intensity it gives for each clause. Although not directly gathered from the weight, we know what clauses are used for inclusion or exclusion in the TM model. With this we apply negation to the weights as appropriate.

For each clause we gather the literals used to build up the conjunctive expressions in propositional logic. We take note of the max-literals hyper-parameter being able to temporary exceed the limit [2]. As such, we take this into consideration so we do not incorrectly represent the influence of each literal.

Overall, to represent this we produce both HTML and LaTeX as output. The visual of both outputs are overall similar. We use the HTML output to more rapidly iterate, while the LaTeX output is used within this paper. The intensity and direction of each words influence is conveyed by the colors red and green, and everything between.

Since there are some differences between the input and what is actually seen by the TM model, we tokenize the input using the same method as when training the model. We do this mainly since we do not want to reimplement the lemmatizer. Additionally, we make a dictionary of all the words and point to where in the input text this is present.

For simplicity sake we use the character offset, as it makes it easier to color in later. A word can map to multiple areas in the input text, this is a side effect of using the binary bag-of-words representation, which discards the word order. Our dictionary accounts for this by being formatted as the type `Dict[str, Set[Tuple[int, int]]]`.

To populate our dictionary we; loop over each word, tokenize it, add the start and end offset. We do not need to care if the word exists already, as we parse the input text sequentially. We also do not care about maintaining the order of where the word offsets are, so we use a set as it has the lowest memory footprint.

When we read out the state of the TM, we can see what literals, or rather what words we should colorize and to what degree. We scale these colors so they are contained within the red to green gradient. With this approach we do not have disproportionately use colors at the edges, but rather uniformly distribute over the whole gradient range.

We colorize each word in the previously created dictionary, but make sure to distribute the intensity of each offset equally to not misrepresent how much a word actually contributes. As an example, in the extreme case of having two unique words, the first one repeated 50 times, and the second word only occurring once. It would be misleading to not weaken the color of the first word, as in the end it takes more area to display the 50 words. All stop words, special characters, and otherwise filtered words are left alone with no color applied, the exact same process as when tokenizing for training. On the other hand it would also be misleading to spread the color so thin it is no longer visible, even though that one word majorly contributes to the classification of the input text. We have considered this and left this to future work, for now we accept the inherent risk in this approach.

An example on how we present the explainable part of a TM can be seen in Figure 3.3. This is not a part of the results, but rather a visual on how we will present the results, along with a color gradient showing the intensity of each color, and what direction it sways the classification. This only represents interpretable results from a model with two classes. You may also notice that some words are not fully colored uniformly, this is an intentional

design decision as to not mistakenly color parts of the word which has not been considered. The highlighted part is the cleaned part of the word, especially important here is ignoring special characters and using the lemmatizer. The darkest red sways the classification the most towards 0, or fake news, while the darkest green sways the classification towards 1, or real news. The color gets lighter the further away it gets from the extremes, until it switches over to grey, which is considered neutral, then to the opposite color.

The grey color is an edge case, and is actually possible to hit. We limit our colors to the colors defined in LaTeX, the name of colors are "green" and "red", the range is limited to 1-100 inclusive, without any decimals. The hidden design detail is that we limit our previously calculated real value to 2 significant digits. With this design it is possible to hit the neutral color, which we assign no value to other than saying it sways the classification in no direction. In reality there might be a slight bias, but the representation in Figure 3.3 is not made to be 100% accurate, rather an alternative way to view the state of the TM model making the classification, and limiting it to the words being classified here. Obsessing over details here would be detrimental to the otherwise gut-feeling given from the colors. You can also see how the same words are colorized the same in the figure. "metre", "oxygen", and "mountain".

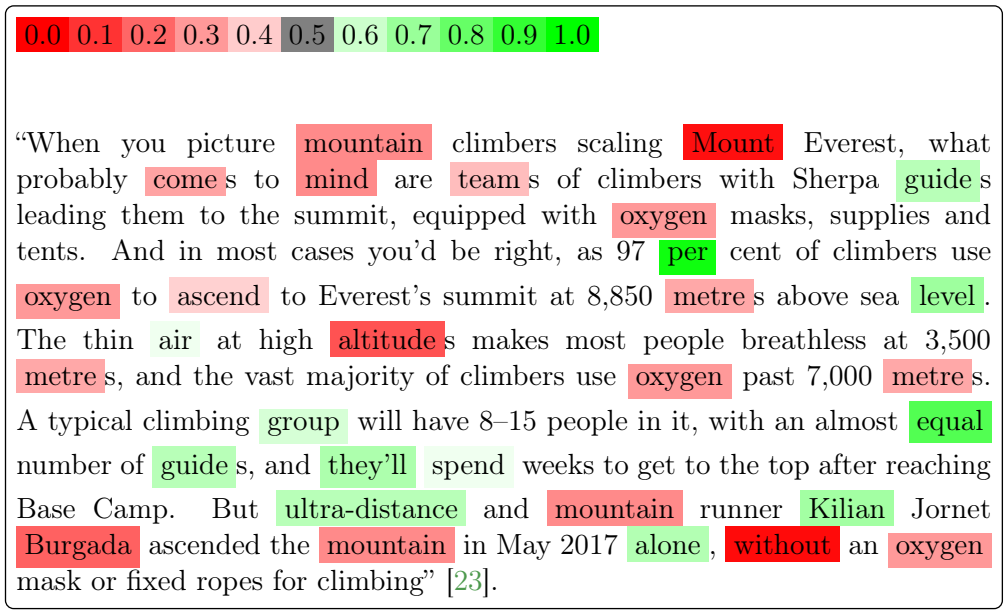

Figure 3.3: Example explainability with weighted coloring. Text is a direct quote [23].

Chapter 4

Results

In this chapter we will go over our numerical results. The implication of every single output is not readily apparent, so we will parse and explain the significance of the results we get. We will bring up considerations made in terms of trade-offs to make it clear when one should worry or when it is safe to ignore.

4.1 Comparing our Results

We want to compare our results against results found in other papers. We will note the differences in terms of preprocessing and postprocessing, and how significant these differences are. We will not compare speed, as these depend highly on the hardware, how many tasks are run in parallel, and the lack of comparable results. We will however take note of relative speed changes, if there is a significant difference. We have omitted some datasets as they do not convey anything useful not covered by the other datasets.

As a baseline, we will consider the performance we see after optimizing for hyper-parameters in Table 4.1. Due to hardware limitations we have not optimized the hyper-parameters for some models, thus omitted them from the table. We found a significant variance in performance when using sub-optimal hyper-parameters. On average, we saw a difference of 11.0% when comparing the worst case with the best case for accuracy scores. Of the results seen in Table 4.1, we found the optimal hyper-parameters as seen in Table 4.2.

Model	PolitiFact				GossipCop			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
TM+Text	0.802	0.819	0.812	0.788	0.745	0.688	0.740	0.698
TM+Domain	0.859	**0.860**	0.843	0.846	**0.839**	**0.848**	0.708	0.730
TM+Text+Domain	**0.863**	0.852	**0.865**	**0.853**	0.818	0.754	**0.744**	**0.742**

Table 4.1: Performance of TM with optimized hyper-parameters. **Bold** denotes largest in column.

Dataset	**Features**	**T**	**s**
PolitiFact	text	150	15
GossipCop	text	250	15
PolitiFact	domain	250	5
GossipCop	domain	250	5
PolitiFact	text, domain	200	15
GossipCop	text, domain	200	15

Table 4.2: Overview of hyper-parameters used for Table 4.1.

For both the PolitiFact and GossipCop datasets, we note that *just* looking at the domain is a $6.7\% - 11.2\%$ improvement in accuracy, and a $4.4\% - 6.7\%$ improvement in F1 score when compared to just using text as input. This tells us that the origin is much more important than the content itself. When combining both the text and domain features, we see a slight but measurable increase in F1 score, 1.2%. We see a noticeable dip of 2.5% in accuracy for the GossipCop dataset. We speculate this is explained by the inbalance in the dataset used, as the F1 score actually shows an improvement over just using the domain to classify. We will use "TM+Text+Domain" when comparing both datasets against other papers. Not all the papers have all metrics available, but we will put focus on accuracy and the F1 score. We drop 0 records from PolitiFact, and 725 records from GossipCop, due to filtering rules outlined in the system chapter. We are not again not particularly worried about class imbalance, as we also consider the F1 score when it is available.

4.1.1 FakeNewsNet

In the FakeNewsNet paper, we are provided a set of metrics [43], seen in Table 4.3. For these we consider see the dataset PolitiFact performs best with "Social Article Fusion", and the dataset GossipCop performs best with "Logic regression". When comparing this against our own results we outperform their results significantly in PolitiFact ($17.2\% - 19.9\%$), and get beat by a decent margin in GossipCop for the F1 score (5.8%), but we beat the SAF in terms of accuracy (2.7%). We do not put much weight on the "win" in terms of accuracy, as the F1 score is more representative of the actual performance.

There are not many details around the preprocessing steps performed over the dataset, which is somewhat expected with the goal of the paper being providing the dataset, and not fine-tuning any kind of model. We do however expect that the process is reasonable enough for the results to be somewhat comparable to to other papers. The authors do mention that they have a 80/20 split on training and testing set. There are no specific details on if some records were left out while preprocessing, but we can safely assume none were removed as this is the paper that introduces the dataset.

Model	PolitiFact				GossipCop			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
SVM	0.580	0.611	0.717	0.659	0.470	0.462	0.451	0.456
Logic regression	0.642	0.757	0.543	0.633	**0.822**	**0.897**	0.722	**0.799**
Naive Bayes	0.617	0.674	0.630	0.651	0.704	0.735	0.765	0.798
CNN	0.629	**0.807**	0.456	0.583	0.703	0.789	0.623	0.699
Social Article Fusion /S	0.654	0.600	0.789	0.681	0.741	0.709	0.761	0.734
Social Article Fusion /A	0.667	0.667	0.579	0.619	0.796	0.782	0.743	0.762
Social Article Fusion	**0.691**	0.638	**0.789**	**0.706**	0.796	0.820	**0.753**	0.785

Table 4.3: Performance from FakeNewsNet paper [43]. **Bold** denotes largest in column.

4.1.2 N-gram, Word Embedding and Topic Models

In this paper [30], we get a different set of metrics to compare against, along with a more complete description of the preprocessing stage, which we will put into perspective of the results we get. We can see the metrics as presented in Table 4.4. Overall, the "N-gram" model performs admirably for both PolitiFact and GossipCop in terms of accuracy and F1 score. "LogReg" does however outperform our proposal over the GossipCop dataset ($0.24\% - 7.8\%$). When compared against the TM model with domain and text features, we see "N-gram" being bested by a significant margin for PolitiFact ($7.3\% - 8.6\%$), and beat by a measurable margin for GossipCop ($0.24\% - 3.8\%$).

Here they use a 80/20 training and testing split. As for preprocessing they use tokenize, lowercase, use stemming, removal of; duplicates, punctuation, special characters and symbols, hash from hashtags, and stop words. This is a reasonable set of actions for preprocessing. After preprocessing they ended up removing 70 records from PolitiFact and 2069 records from GossipCop. While this compares nicely with our preprocessing steps, we do have concerns around the records removed from the GossipCop dataset, and would preferably rerun these tests to confirm the results.

Model	PolitiFact				GossipCop			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
LogReg	0.64	0.76	0.54	0.63	**0.82**	**0.90**	0.72	**0.80**
Social Article Fusion	0.69	0.64	**0.79**	0.71	0.80	0.82	0.75	0.79
N-gram	**0.80**	**0.79**	0.78	**0.78**	**0.82**	0.75	**0.79**	0.77
Topic	0.60	0.55	0.53	0.51	0.51	0.51	0.51	0.47
Word2Vec	0.73	0.73	0.74	0.73	0.78	0.71	0.76	0.72
N-gram + Topic	0.77	0.76	0.76	0.76	**0.82**	0.75	0.78	0.76
N-gram + Word2Vec	0.72	0.72	0.73	0.72	0.78	0.71	0.76	0.72
Topic + Word2Vec	0.42	0.49	0.49	0.39	0.63	0.60	0.64	0.60
N-gram + Topic + Word2Vec	0.40	0.45	0.48	0.36	0.58	0.57	0.60	0.54

Table 4.4: Performance from N-gram paper [30]. **Bold** denotes largest in column.

4.1.3 SpotFake+

SpotFake+ is a multimodal model, meaning it in addition to NLP, it also another form of input. SpotFake+ specifically takes images as input [44]. We can still use this for comparison, both results from a variety of single-modal modals and multi-modal are present, which we will compare our results again. First off, let us consider the text-only models in Table 4.5. This paper only provides the accuracy metric, so we will solely use that for our comparison. For the PolitiFact dataset the TM model significantly outperforms "XLNet + dense layer" by 14.3%, and for the GossipCop dataset "XLNet + CNN" beats out our TM model by 2.7%. The authors did not go into significant detail for the preprocessing stage, but from the achieved accuracy scores we can consider them reasonable enough for comparing against our own model.

Going further, we can compare the the results found in the multimodal models. Even if this is not a fair comparison for the TM models, as the multimodal model is augmented with images, while the TM model only has access to the article content and domain. In Table 4.6 we again see the achieved accuracy from the different models presented in the paper. Even with the additional detail available to this model, our TM outperforms "SpotFake+" by a decent margin for PolitiFact, 2.0%. As for GossipCop, we see the "EANN" model take the lead by a decent margin over TM, 5.1%.

The paper provides a brief overview of the steps taken during preprocessing of the multimodal model. There is a lack of details around the text only models, but this is to be expected when the paper mostly focuses on multimodal models. We can still use the text models in our comparison, even if the particular steps are missing. Preprocessing steps for the image models are laid out, although brief; logos were removed, and samples without images were dropped. After preprocessing they ended up removing 571 records from PolitiFact and 9300 records from GossipCop. This is a substantial amount of articles that were removed, which may skew the results more than expected, but we can not say for certain since only the accuracy score is available.

Model	PolitiFact	GossipCop
	Acc	Acc
SVM	0.58	0.497
Logistic Regression	0.642	0.648
Naive Bayes	0.617	0.624
CNN	0.629	0.723
SAF	0.691	0.689
XLNet + dense layer	**0.74**	0.836
XLNet + CNN	0.721	**0.84**
XLNet + LSTM	0.721	0.807

Table 4.5: Performance from SpotFake+ paper, text model [44]. **Bold** denotes largest in column.

Model	PolitiFact	GossipCop
	Acc	Acc
EANN	0.74	**0.86**
MVAE	0.673	0.775
SpotFake	0.721	0.807
SpotFake+	**0.846**	0.856

Table 4.6: Performance from SpotFake+ paper, multimodal model [44]. **Bold** denotes largest in column.

4.1.4 Heuristic-driven Uncertainty

In the paper [12] we see an approach which achieves better overall performance, 91.56% overall in both accuracy and F1 score. We speculate this significantly higher accuracy is achieved through a better preprocessing method specifically tuned for the FakeNewsNet dataset, along with augmenting the dataset with externally scraped data. The dataset was split into 80% training, 10% testing, and 10% validation.

The Python library "tweet-preprocessor" was used to clean, tokenize, and parsing of tweets. This includes URLs, hashtags, mentions, reserved words, emojis, and smileys. This requires additional data not present in the standard FakeNewsNet dataset, such as scraping all the referenced tweets. A set of different state-of-the-art tokenizers available from HuggingFace[1] was used. Outside of tweets, articles were filtered for usernames, and URLs, the specifically mentioned ones include Instagram, Facebook, and Twitter, but others may also be filtered out. The vocabulary was trained from external knowledge, specifically mentioned sources include GLUE, wikitext-103, and CommonCrawl. Transfer learning was used for a variety of pre-trained language models. Additional prediction vectors were augmented the dataset with NewsBERT, by using the article body.

The heuristic part of the paper refers to the final step in the overall process. With that said, the combination of all these methods make this paper not directly comparable due to the sheer amount of new and external data which is not available even indirectly from FakeNewsNet. While this heuristic-driven model outperforms all other models presented for comparison, it is no longer a direct comparison of models working with the FakeNewsNet dataset. Here the goal is rather to make a more accurate model, we can see the appeal of this, but this is not well suited for comparing models directly, rather, the paper may be used as a way to improve the preprocesing stage, but this is out of scope of this paper.

Model	Acc	Prec	Rec	F1
FakeFlow	0.82	0.82	0.82	0.82
One-Hot LR	0.7670	0.7670	0.7670	0.7670
FakeNewsTracker	0.7186	0.7186	0.7186	0.7186
Ensemble Model + Heuristic Post-Processing	0.9007	0.9007	0.9007	0.9007
SFFN (with MCDropout) + Heuristic Post-Processing	**0.9156**	**0.9156**	**0.9156**	**0.9156**

Table 4.7: Performance from the Heuristic-driven Uncertainty paper [12]. **Bold** denotes largest in column.

[1] https://github.com/huggingface/tokenizers

Model	p=.25 acc	p=.75 acc	p=.25 f1	p=.75 f1
Election Day	0.7782	0.8759	0.6089	0.6758
FakeNewsAMT*	0.4688	0.5000	0.4588	0.4891
Celebrity*	0.7200	0.7300	0.7182	0.7238

Table 4.8: Comparing drop-clause on TM with hyperparameters $T = 200, s = 15, clauses = 10000$.

4.2 Effects of Drop-Clause TM Variant

From Table 4.8 we see a clear trend regardless of dataset in terms of accuracy and F1 score. For "Election Day" we see a significant increase in both accuracy and F1 score when increasing the amount of dropped clauses from 25% to 75%. While we expected an increase in performance [40], we did not expect such a significant increase. As for comparison sake, we include 2 additional datasets, FakeNewsAMT*, and Celebrity*. The differentiating factor for these two is that the TM parameters are optimized for the "Election Day" dataset. While these hyper-parameters works decently, we use this as an example to demonstrate the significance of searching for the correct hyper-parameters.

*Hyper-parameters for these models were not tuned. This is to show that the drop-clause feature is not dependant on whether the models were optimized for the given dataset or not.

Outside of accuracy improvements, we can see a significant speed improvement when training the model. This is true for all the tested datasets, and is an expected outcome by increasing the drop clause percentage from 25% to 75% [40]. It should be noted that we do not see any disadvantages of implementing the drop-clause TM variant. From the observed results we only see positives of this approach. We will consider degradation of explainability in the explainability section, we do not expect this change to affect it, but we found it important to highlight as a part that has been considered.

4.3 Effects of Max-Literals TM Variant

From the A model in Table 4.9 we see a small but measurable decrease in both accuracy and F1 score. This decrease is to be expected [2], as we operate with much simpler propositional logic which is a desirable feature for explainability. In the extreme case of limiting to 3 literals we see a more significant decrease in accuracy. However, when shifting our focus to the F1 score we see this is not much of a concern, and more of an unfortunate run resulting in the fall in accuracy. This is mostly caused by the small size of the dataset, as larger datasets do not have this much variance. With the F1 score we see a steady decrease.

While it may sound contradictory, lowering the max-literals is not always desirable, even if we disregard the accuracy penalty. We only want lower max-literals when presenting the propositional logic directly. On the other side, if we want to parse out and quantify the significance of each token in the form of a color gradient, we want as many literals as possible. This is to have more granular shades to colorize the text, highlighting which words contribute towards each classification. We go into further details around this in the explainability section.

The "Election Day" dataset consists of approximately 1300 rows. Additionally, from the B model in Table 4.9 we see a more steady score progression. In this case when using a different set of hyper-parameters we actually see the opposite extreme in terms of accuracy. With the max-literals set to 32 we see a significant jump in accuracy.

Model	Accuracy	Precision	Recall	F1
"A" Election Day max-literals=3	0.7932	0.6005	0.6878	0.6149
"A" Election Day max-literals=8	0.8346	0.6198	0.6780	0.6381
"A" Election Day max-literals=16	0.8459	0.6104	0.6350	0.6204
"A" Election Day max-literals=32	0.8459	0.6250	0.6678	0.6407
"B" Election Day max-literals=3	0.8346	0.6198	0.6780	0.6381
"B" Election Day max-literals=8	0.8459	0.6315	0.6843	0.6499
"B" Election Day max-literals=16	0.8496	0.6294	0.6699	0.6448
"B" Election Day max-literals=32	0.8835	0.6677	0.6395	0.6516

Table 4.9: Metrics given TM with hyper-parameters;
A: $T = 150, \quad s = 10, \quad clauses = 5000,$
B: $T = 200, \quad s = 15, \quad clauses = 10000.$

4.4 Explainability

As a large part of the TM, we can use the internal state to represent a classification outside of the output itself. One such example is in Figure 4.1, where the colors red and green are used to show the weight of each word. The more intense a word is, the more it sways the word towards that classification. It is largely impossible to evaluate if the explanation is right or wrong, but this gives some insight into what influeses the model the most. One thing that may be lost in translation is the distribution of weight across repeated words, for headlines like in Figure 4.1 this is not an issue, but for article bodies it may. The TM itself does not distinguish if some input contains one instance of the word, or 200. For the explanaibility color view we must handle it differently, as we are trying to represent repeated words weaker overall, as they take up more area as text than when in a bag-of-words representation. As such, the colors may become much weaker for longer texts.

In any case, in the figure we see two words swaying the classification towards real news, and 3 words swaying the classification towards fake news. The other words are either stop words or not part of the vocabulary. For the word "Video", we see it sway the classification towards real news, with an intensity of 32%, this seems reasonable. The second word, "shows" only has part of the word colorized. This is caused by the lemmatizer, and is represented as such to not cause confusion to what part was evaluated, and overlaps better with the actual vocabulary used. The word shows as a 15% red, which by gut feeling seems correct, considering all the "Study shows" articles which are fake. "Nancy" shows as 16% green. "Pelosi" is however 9% red, the conclusion we can gather from this is that the dataset either contained a lot of references to "Nancy" which did not map up to the same person as "Pelosi", or that fake news tend to only use "Pelosi". Lastly, "arrested" shows as 55% red. This is a simple aggregate representation of the TM, which does not properly convey the sub patterns that it may have learned and are present here.

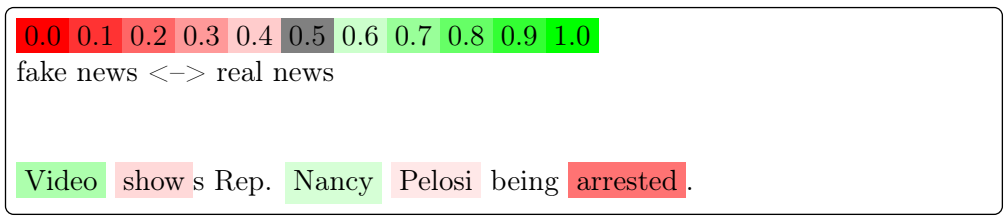

Figure 4.1: Example explainability taken from TM model.

Chapter 5

Discussion

In this chapter we will go over the things that were planned, but for some reason or another did not work out as expected. We will also cover work that was done, but ultimately did ultimately not benefit this thesis in a meaningful manner.

5.1 Limitations

In the preprocessing stage we assume the dataset is in English. Dependants of this include the tokenizer, stop words, and lemmatizer. The tokenizer expects spaces to separate words, so CJK or similar languages would not tokenize correctly under the current configuration. This is an intentional design decision to limit work into areas which we are not too familiar with. Additionally, we expect the additional work going into adding multilingual support would not hold back our results in a meaningful way.

The TMU library (v0.8.1) does not support seeding the PRNG which results in the training stage to not be reproducible. We expect CPU training is possible to be made fully deterministic, given the same hardware. This is mostly due to hardware differences, such as with floating point precision and double floating point precision. We also have concerns about differences due to platform or driver specific quirks. GPU vendors have been notoriously known for cheating in video game benchmarks, which we fear might transfer over to ML. We are aware of the randomness introduced by the shuffle argument to TMU's fit function, this is already accounted for by seeding NumPy. The randomness we can not seed is however expected to be related to the drop-clause TM variant.

The current code does not support online learning, as it was considered of little to no use when we only compare models at a fixed epoch. We need stable points to compare against, otherwise we would have to train the model to exhaustion, which necessitates additional hardware. This also introduces an additional variable, exactly when is a model "done"?

5.2 GPU

The system as described assumed the host machine has an Nvidia GPU. This is a limitation carried over from the TMU library. The provided code has no limitations on the Graphics Processing Unit (GPU), so if TMU adds support for AMD's ROCm or Intel's oneAPI, it should have no issues running under those environments. Guarantees can however not be made around the deterministic nature of the code. Adding an additional abstraction layer may not behave exactly the same down to the implementation details, especially across vendors. If support for ROCm was added, this would add support for AMD cards. If

support for oneAPI was added, we expect support for Intel, AMD, and Nvidia to be available through this framework. With this we expect a lower probability of deterministic behavior being possible for TMU.

5.3 Authority

With the target domain in mind, fake news, trusting the classification of the model is equally as important, if not even more. When a user study was carried out in a related domain, fake reviews[21], it was found that users will largely not trust both the classification and the reasoning behind the classification over their own conclusion. Since this study was based on fake reviews, we can consider it as politically neutral and not something to be concerned about in this regard. This tells us the main issue is distrust in the model or presentation overall, and not an issue with distrust derived from the source, but rather the technology itself. The study also pointed out that users did not stray away from their initial decision to a measurable degree when the output of the model, and the accompanying explanation was made available. In combination with the fact that humans have a tendency to only be 4% better at distinguishing fake from real news [8], it is somewhat shocking that the underlying distrust in the model or it's explanation is still present.

5.4 Saving the state of TM

Saving the state of the TM model was a planned feature. After implementing the feature we saw a huge drawback to just picking the TM model. Picking is a Python term to save a Python object to disk, it does not work on every object, but TMU had to specifically add support for picking the TM model. When picking the model we found that for a dataset of 500 entries and 4922 features resulted in a 630 MiB file. This was further compressed to 80 MiB, but the overall size was too large to be feasible. This is a relatively small dataset, so the size would only grow from here.

If we were to save every state of the TM we would add significant overhead, and have no practical way to easily distribute the model. Take the hyper-parameter tuning as an example, here we run 100s of tests over 150 epochs each. This would end up with $200 \times 150 \times 630 MiB \approx 18 TiB$. While we can choose to just overwrite the file for each epoch, we would run into other issues, such as write speed, SSD wear, and HDD corruption. For an SSD this would consume 3% of the total possible writes, assuming a consumer level SSD. While HDDs do not have the same wear concerns, at these sizes we need to consider Mean Time Between Failure (MTBF), which in terms of HDDs is how many bytes needs to be written before a bit is flipped. For consumer HDDs this is in the 1000s of GiB, which is low.

5.5 Trainining on More Features

A lot of the datasets included references to source tweets by their ID, such as the FakeNewsNet dataset. Loading and training on this ID was attempted, but ended up not affecting the results at all. This was mistakenly interpreted as being the user ID, and not the tweet ID. Since tweet IDs are unique we neither gained or lost any accuracy from adding these as additional features. To benefit from training on anything related to the tweet ID, we would have to request the corresponding user ID for each tweet, assuming the tweets are still available. This is unlikely to be something useful to expand on, as fake news are more likely to be removed, skewing the class balance. Using the relationship between users and what articles they share, if available in full, is expected to contribute to a better model. Because of how the tweet IDs were formatted, and there was an abundance of IDs it increased the time in the preprocessing stage by approximately 20% on the FakeNewsNet dataset.

Chapter 6

Conclusion

We set out to improve the accuracy of fake news detection with the help of TM, with particular focus on recent improvement such as with the drop-clause and max-literals features. In large parts we have achieved this goal, with a measurable improvement with the drop-clause and max-literals features. We tested the drop-clause feature with the amount of dropped clauses set to 25% and 75%, the standard TM in comparison would be equivalent to 0%. Over multiple datasets we found the F1-score increased between 0.78% to 10.99% with the mean of 6.12%. The max-literals feature was also tested with the parameters 32, 16, 8, and 3. This gives us the F1-scores 65.2% 64.5%, 65.0%, and 63.8% respectively. From this we note that there is a clear trend of decreasing accuracy, which mirror the findings found in the original proposal for the max-literals feature.

A focus on deterministic code was set early on, as such both preprocessing and postprocessing are fully deterministic. Sadly, due to the way the TMU works it is not possible to properly seed this part of the process. There was also a focus on explainability. As it has been shown time and time again, humans are laughably bad at distinguishing fake from real news, barely 4% better than chance. As such, we felt it was important to properly convey the reasoning behind the classification given. This comes in the form of colorized text, where the color represents the intensity and sway of each word.

6.1 Future Work

Our results are directly compared against results presented in other papers. Ideally we would reimplement these ML models, as to guarantee the preprocessing stays the same. While the description of the preprocessing stage seems reasonable, we would like to compare the models rather than the quality of the preprocessing. We have noted down the differences and how these may affect the performance of the model, in the results chapter. While this may ultimately not matter, we believe it would be a decent addition for reducing uncertainty. We put a focus on using well-known fake news datasets, this was decided upon because we decided against reimplementing the different ML models. With more niche datasets at our disposal, we may find different results, which would be interesting to explore. Currently the datasets we use are limited to English fake news, but branching out, while it might present different challenges, the TM would have more to work with if we use the same parameters. Another area we did not explore was other languages. If we want to expand this to other languages we need to reconsider parts of the code, as some assumptions were made. The preprocessing stage relies on English stop words and an English lemmatizer, and a tokenizer splitting on spaces. This is an effective combination when solely working on English content, but not so much outside of that. The tokenizer will have issues with CJK languages, as they do not contain spaces. Since we are not familiar or had any focus on any of the CJK

languages we are unaware of any libraries that can assist in tokenizing in the same manner as we use here. A possible remedy would be using a n-gram tokenizer, which can work on individual characters, instead of whole words.

We have walled off our research to the fake news domain. While this is a subdomain of the larger NLP field, we believe the findings presented herein applies to other NLP tasks as well. Not only limited to adjacent domains, such as fake reviews or spam detection, but any NLP task that may benefit from the reason behind the classification. Of course, we have also shown increase in accuracy over state of the art, and TM could be beneficial for this alone. The explainable part could also use some work to make it more intuitive. While the presentation described in this thesis is *an* approach, there is more work to make it overall more accessible. The simplest approach is resolving each literal and presenting this to the user, which would rely heavily on the max-literals TM feature, as to not make the expressions overwhelming. Something more advanced would be further processing these words, such as shown in this thesis, or turn the logical expressions into full sentences. Explainability is a complex thing to represent in a meaningful manner, so more research into this area always desirable. Particularly, complexity is harder to interpret than simple repetition [22]. We would like to work further on how to colorize the text, such as with longer texts. Here we would like to overcome parts of the bag-of-words representation the TM is limited to. We can aggregate the classification over each sentence, or a rolling window of tokens. Since inference is cheap, this may give a better understanding of what parts are more likely to be fake news, and which parts are benign. This could also help give a more granular overall classification.

Bibliography

[1] K Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "Extending the tsetlin machine with integer-weighted clauses for increased interpretability." In: *IEEE Access* 9 (2021), pp. 8233–8248.

[2] K Darshana Abeyrathna et al. "Building Concise Logical Patterns by Constraining Tsetlin Machine Clause Size." In: *arXiv preprint arXiv:2301.08190* (2023).

[3] Hunt Allcott and Matthew Gentzkow. "Social media and fake news in the 2016 election." In: *Journal of economic perspectives* 31.2 (2017), pp. 211–236.

[4] Julio Amador, Axel Oehmichen, and Miguel Molina-Solana. "Characterizing political fake news in twitter by its meta-data." In: *arXiv preprint arXiv:1712.05999* (2017).

[5] Geir Thore Berge et al. "Using the Tsetlin Machine to learn human-interpretable rules for high-accuracy text categorization with medical applications." In: *IEEE Access* 7 (2019), pp. 115134–115146.

[6] Bimal Bhattarai, Ole-Christoffer Granmo, and Lei Jiao. "Explainable Tsetlin Machine Framework for Fake News Detection with Credibility Score Assessment." In: *the 13th Conference on Language Resources and Evaluation*. 2022, pp. 4894–4903.

[7] Bimal Bhattarai, Ole-Christoffer Granmo, and Lei Jiao. "Explainable tsetlin machine framework for fake news detection with credibility score assessment." In: *arXiv preprint arXiv:2105.09114* (2021).

[8] Charles F Bond Jr and Bella M DePaulo. "Accuracy of deception judgments." In: *Personality and social psychology Review* 10.3 (2006), pp. 214–234.

[9] Alexandre Bovet and Hernán A Makse. "Influence of fake news in Twitter during the 2016 US presidential election." In: *Nature communications* 10.1 (2019), p. 7.

[10] Yimin Chen, Nadia K Conroy, and Victoria L Rubin. "News in an online world: The need for an "automatic crap detector"." In: *Proceedings of the Association for Information Science and Technology* 52.1 (2015), pp. 1–4.

[11] K Darshana Abeyrathna et al. "The regression Tsetlin machine: a novel approach to interpretable nonlinear regression." In: *Philosophical Transactions of the Royal Society A* 378.2164 (2020), p. 20190165.

[12] Sourya Dipta Das, Ayan Basak, and Saikat Dutta. "A heuristic-driven uncertainty based ensemble framework for fake news detection in tweets and news articles." In: *Neurocomputing* 491 (2022), pp. 607–620.

[13] Jeffrey Dastin. "Amazon scraps secret AI recruiting tool that showed bias against women." In: *Ethics of data and analytics*. Auerbach Publications, 2018, pp. 296–299.

[14] Jenna Drenten and Lauren Gurrieri. "Crossing the# bikinibridge: Exploring the role of social media in propagating body image trends." In: *The Dark Side of Social Media*. Routledge, 2017, pp. 49–70.

[15] John W Fritch and Robert L Cromwell. "Evaluating Internet resources: Identity, affiliation, and cognitive authority in a networked world." In: *Journal of the American Society for Information science and Technology* 52.6 (2001), pp. 499–507.

[16] Sondre Glimsdal and Ole-Christoffer Granmo. "Coalesced multi-output tsetlin machines with clause sharing." In: *arXiv preprint arXiv:2108.07594* (2021).

[17] Ole-Christoffer Granmo. "The Tsetlin Machine–A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic." In: *arXiv preprint arXiv:1804.01508* (2018).

[18] Ole-Christoffer Granmo et al. "The convolutional Tsetlin machine." In: *arXiv preprint arXiv:1905.09688* (2019).

[19] Richard L Kaplan. "Politics and the American press: The rise of objectivity, 1865-1920." In: *Canadian Journal of Communication* 28.2 (2003).

[20] Ian Kelk et al. "Automatic Fake News Detection: Are current models "fact-checking" or "gut-checking"?" In: *Proceedings of the Fifth Fact Extraction and VERification Workshop (FEVER)*.

[21] Jeonghwan Kim et al. "Can You Distinguish Truthful from Fake Reviews? User Analysis and Assistance Tool for Fake Review Detection." In: *Proceedings of the First Workshop on Bridging Human–Computer Interaction and Natural Language Processing*. Online: Association for Computational Linguistics, Apr. 2021, pp. 53–59. URL: https://aclanthology.org/2021.hcinlp-1.9.

[22] Isaac Lage et al. "An evaluation of the human-interpretability of explanation." In: *arXiv preprint arXiv:1902.00006* (2019).

[23] LearnEnglish. *A biography of Kilian Jornet*. https://learnenglish.britishcouncil.org/skills/reading/c1-reading/biography-kilian-jornet. Accessed June 9, 2023. Sept. 2021.

[24] Jie Lei et al. "From arithmetic to logic based AI: a comparative analysis of neural networks and tsetlin machine." In: *2020 27th IEEE international conference on electronics, circuits and systems (ICECS)*. IEEE. 2020, pp. 1–4.

[25] Zachary C Lipton. "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." In: *Queue* 16.3 (2018), pp. 31–57.

[26] Yiheng Liu et al. "Summary of chatgpt/gpt-4 research and perspective towards the future of large language models." In: *arXiv preprint arXiv:2304.01852* (2023).

[27] Binny Mathew et al. "Hatexplain: A benchmark dataset for explainable hate speech detection." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 17. 2021, pp. 14867–14875.

[28] Raha Moraffah et al. "Causal interpretability for machine learning-problems, methods and evaluation." In: *ACM SIGKDD Explorations Newsletter* 22.1 (2020), pp. 18–33.

[29] OpenSources. *OpenSources*. http://www.opensources.co/. Accessed February 13, 2019.

[30] Oluwafemi Oriola. "Exploring N-gram, word embedding and topic models for content-based fake news detection in FakeNewsNet evaluation." In: *Int J Comput Appl* 975 (2021), p. 8887.

[31] Felipe Oviedo et al. "Interpretable and explainable machine learning for materials science and chemistry." In: *Accounts of Materials Research* 3.6 (2022), pp. 597–607.

[32] Judea Pearl. "Causal inference in statistics: An overview." In: (2009).

[33] Gordon Pennycook and David G Rand. "The psychology of fake news." In: *Trends in cognitive sciences* 25.5 (2021), pp. 388–402.

[34] Verónica Pérez-Rosas et al. "Automatic Detection of Fake News." In: *International Conference on Computational Linguistics (COLING)* (2018).

[35] Sebastian Ruder et al. "Transfer learning in natural language processing." In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*. 2019, pp. 15–18.

[36] Rupsa Saha, Ole-Christoffer Granmo, and Morten Goodwin. "Mining interpretable rules for sentiment and semantic relation analysis using Tsetlin machines." In: *Artificial Intelligence XXXVII: 40th SGAI International Conference on Artificial Intelligence, AI 2020, Cambridge, UK, December 15–17, 2020, Proceedings 40*. Springer. 2020, pp. 67–78.

[37] Rupsa Saha, Ole-Christoffer Granmo, and Morten Goodwin. "Using Tsetlin machine to discover interpretable rules in natural language processing applications." In: *Expert Systems* 40.4 (2023), e12873.

[38] Rupsa Saha et al. "A relational tsetlin machine with applications to natural language understanding." In: *Journal of Intelligent Information Systems* (2022), pp. 1–28.

[39] Gautam Kishore Shahi and Durgesh Nandini. "FakeCovid–A multilingual cross-domain fact check news dataset for COVID-19." In: *arXiv preprint arXiv:2006.11343* (2020).

[40] Jivitesh Sharma et al. "Drop clause: Enhancing performance, interpretability and robustness of the tsetlin machine." In: *arXiv e-prints* (2021), arXiv–2105.

[41] Qiang Sheng et al. "Zoom Out and Observe: News Environment Perception for Fake News Detection." In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 4543–4556. DOI: 10.18653/v1/2022.acl-long.311. URL: https://aclanthology.org/2022.acl-long.311.

[42] Kai Shu et al. "Fake news detection on social media: A data mining perspective." In: *ACM SIGKDD explorations newsletter* 19.1 (2017), pp. 22–36.

[43] Kai Shu et al. "Fakenewsnet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media." In: *Big data* 8.3 (2020), pp. 171–188.

[44] Shivangi Singhal et al. "Spotfake+: A multimodal framework for fake news detection via transfer learning (student abstract)." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 10. 2020, pp. 13915–13916.

[45] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[46] Maciej Szpakowski. "Fake news corpus." In: *URL: https://github.com/several27/FakeNewsCorpus* 27 (2018).

[47] Ivan Y Tyukin et al. "Demystification of few-shot and one-shot learning." In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–7.

[48] Joseph E Uscinski and Ryden W Butler. "The epistemology of fact checking." In: *Critical Review* 25.2 (2013), pp. 162–180.

[49] Adrian Wheeldon et al. "Learning automata based energy-efficient AI hardware design for IoT applications." In: *Philosophical Transactions of the Royal Society A* 378.2182 (2020), p. 20190593.

[50] Adrian Wheeldon et al. "Low-latency asynchronous logic design for inference at the edge." In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 370–373.

[51] Adrian Wheeldon et al. "Tsetlin machine: a new paradigm for pervasive AI." In: *Proceedings of the Scona Workshop at Design, Automation and test in Europe (date)*. 2020.

[52] Xue Ying. "An overview of overfitting and its solutions." In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022.