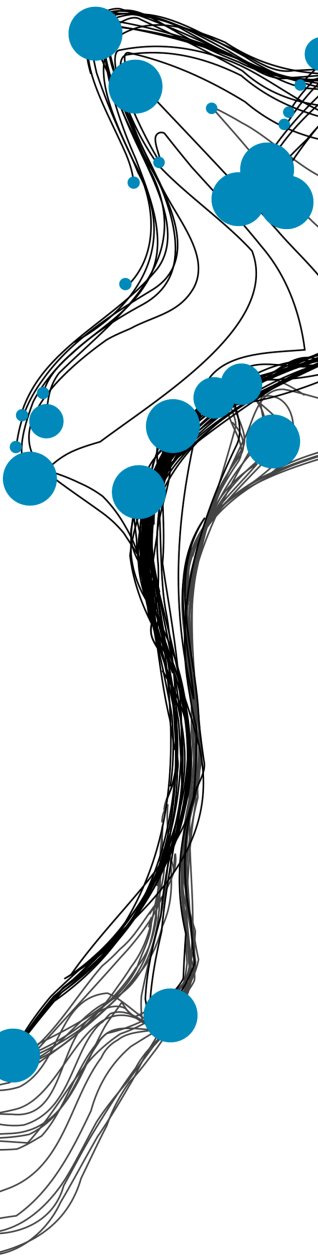


Sentiment as a ground truth stance indicator against fake news

Designing systems for fighting fake news on the Web
a Creative Technology graduation project



UNIVERSITY OF TWENTE.

Emiel Steegh
s1846388

Supervisor: dr. Andreas Kamilaris
Critical Observer: dr. Job Zwiers

February 12, 2021

Contents

Figures	3
Tables	3
Abstract	4
I. Introduction	4
A Situation	4
B Definition	4
C Challenge	4
D Problem statement	5
II. State of the Art	5
A Detection methods	5
B Primary actors	5
C Handling fake news	6
D Shortcomings	6
E Next steps	6
III. Methods & Techniques	7
A The Creative Technology Design Process	7
B Prototyping tools	7
IV. Ideation	8
A Divergence	8
B Convergence	9
V. Specification	9
A Early requirements	9
B A representation of truth	10
C The dataset	11
VI. Realisation	12
A Lab 01 – Sentiment analysis	12
B Lab 02 – Triple extraction	12
C Lab 03 – Sentiment model	13
D Lab 04 – Sentiment pre-processing	13
E Lab 05 – Recurrent neural network	15
F Lab 06 - Snopes Improvements	16
G Lab 07 – Sentiment & Topic	16
H Lab 08 – Model combination	19
VII. Evaluation	20
A Requirements revisited	20
B F1-score	20
C Case examination	20
VIII. Conclusions	20
IX. Future Work	20
Acknowledgements	21
References	21
Appendix	23
A Appendix Lab 01 - Sentiment analysis	23
B Appendix Lab 02 - Triple extraction	26
C Appendix Lab 03 - Sentiment model	28
D Appendix - Article module	30
E Appendix Lab 04 - Sentiment pre-processing	32
F Appendix Lab 05 - Recurrent neural network	36
G Appendix Lab 06 - Snopes improvements	47

H	Appendix Lab 07 - Sentiment & topics	49
I	Appendix - PyLDAvis	57
J	Appendix Lab 08 - Sentiment & topics	58

LIST OF FIGURES

Figures

1	Characteristics of the fake news genre	4
2	Simplified process: anti-fake news	5
3	Design Process for Creative Technology	7
4	Knowledge extraction and comparison	10
5	Example of an SPO	10
6	Triple extraction and comparison model	13
7	Sentiment sequences of 4 fake (red) and 4 real (green) articles .	14
8	Up- and downsampling a sequence (from 30 to 100 and 10) . .	14
9	Simple illustration of the vanishing gradient problem	15
10	Final LSTM-model design	16
11	Sentiment & topic knowledge base creation architecture	17
12	Text preprocessing method	17
13	Additional steps during first time text preprocessing	18
14	Final knowledge base creation model	19

LIST OF TABLES

Tables

I	Stakeholders of a fake news combatant	9
II	Model requirements	10
III	Reviewed available datasets	11
IV	sources of sentiment tools	12
V	VADER applied to modified sentences	12
VI	Weka J48 (50 leaves) results of topic and avg. sentiment data . .	19
VII	Confusion matrix final model	20

ABSTRACT

This report investigates a novel ground truth approach to fake news detection, focusing on sentiment detection to interpret the author's stance. A three-pronged model was built combining an LDA for topic detection, VADER to create sentiment sequence, a stacked LSTM to interpret those sentiment sequences and a statistical description of the sequences. The model interpreted the popular ISOT fake news dataset to build a knowledge base and achieved an 80% instance classification accuracy on the dataset using J48, finding value in sentiment analysis as a tool for fake news detection.

Keywords— fake news, ground truth, sentiment analysis, stance detection

I. INTRODUCTION

A. Situation

Reliable information is essential. Currently, COVID-19 poses a threat to the health of the human race. Dealing with the spread of the virus has become a vital issue of our time. However, for the public to adequately respond to the tasks ahead, they need to be well informed. The internet is the most significant public source of information, but it faces an infodemic[1].

Trust in mainstream media is at an all-time low. There is a staggering amount of information available online, and not everything on the internet is truthful. People are now responsible for deciding what they accept as true, a time-consuming and challenging task. It is easier to accept the first article available than looking for multiple sources to establish a well-rounded view. Low media literacy, which is especially prevalent in areas with lower socioeconomic status[2], leads to quicker acceptance of misinformation and disinformation.

Content creators can manipulate information for monetary or ideological interest. In the 2016 United States elections, fake news has had a measurable impact on voter behaviour. Weaponizing false information to sway elections or spread doctrine is a dangerous practice. The other main incentive to create and spread fake news is money. The internet has a vast ad-based economy, anything that attracts attention has value. Sensationalist fake news spreads much faster and garners more view than the average truthful article[3].

Manipulating elections is rather obviously undemocratic and therefore contrary to human rights as described by the United Nations[4]. Fake news can cause civil unrest or fuel hatred, as in the case of India and Pakistan[5]. Nevertheless, disinformation with profit as a goal can be just as dangerous. In the case of COVID-19, for example, viral news of alternative treatments have put human lives in danger[6].

B. Definition

There is no single widely accepted definition for the genre of fake news. [7, 8] Agree in their definition, arguing for three pillars: Fake news is not veracious, presents itself as news and is intentionally deceitful. These definitions exclude satire content like the onion, unintentional mistakes, parody and advertising. [9] On the other hand, maintains that satire and humour also constitute fake news, but are not deceitful. This body of research will use Egelhofer's well-argued and summarized definition for the fake news genre described by Figure 1.

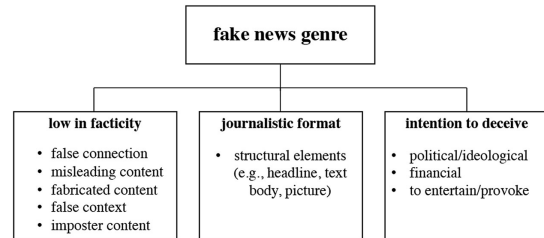


Fig. 1: Characteristics of the fake news genre

C. Challenge

In [10] Lessig identifies four forces that regulate our actions: law, social norms, the market and architecture. The fake news problem is approachable through all four forces[9]. Policies might change the legality of creating fake news and help regulate news environments. Social norms can shift in paradigm building trust in reputable sources and enhancing media literacy. The market can tackle the financially driven fake news by changing the valuation of attention. Not necessarily these responses, but at least the pillars, are all part of the solution, and to effectively control the problem each needs to be addressed. This research will focus on the architecture domain, building constraints for the web to tackle the existing fake news. It will support a short-term implementable

response that can become part of the long-term solution.

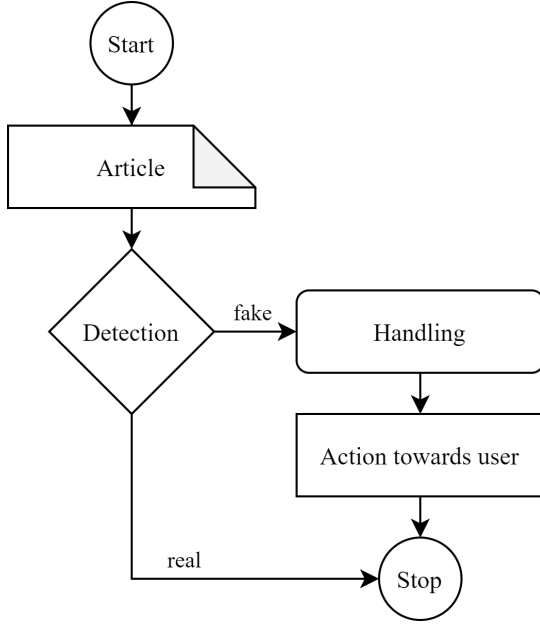


Fig. 2: Simplified process: anti-fake news

D. Problem statement

The process of dealing with fake news on the web has two core steps: detecting and handling. Figure 2 shows a simplified process of this approach. Further on in this work the model will be made more concrete and highlight the point of focus. The decision for the next step will be based on the state of the art analysis in the next chapter. The analysis will allow us to specify the question:

"How can we improve the existing systems against fake news?"

II. STATE OF THE ART

The previous chapter establishes the fake news problem. And, there is a world-wide need for a solution to this problem. The following chapter outlines the state of the art solutions by creating a taxonomy for different existing or theorized approaches that aim to detect and deal with disinformation online. It will suggest where to direct scientific efforts to be able to keep information online reliable. In doing so, it will function as a starting point for research on combatting fake news and offer an insight into the building blocks of previous types of work.

A. Detection methods

There are two main branches of detection methods in deciding if a piece is fake or not; computational

and crowd-sourced. The types of input can be discerned as content cues and contextual cues.

The computational branch attempts to detect whether something might be fake news automatically. The inputs used involved in this process are the content, and the context of the text [11]. Using natural language processing on the body of text allows computers to find linguistic cues hinting at non-authenticity. This language leakage presents itself in the form of linguistic features, for example; grammar, vocabulary, quantity and specificity/expressiveness observed by [12]. Neural networks use these features to train to give confidence scores of whether a text is deceptive [13, 14].

The contextual features that provide neural networks with information for the discrimination of fake news include; user data and interactions, propagation on the network and linked data (e.g. sources on the same topic)[11, 15, 16]. M. Della et al. [17] examine the good results of context-only approaches but observes shortcomings when there are only a few interactions. Combining contextual cues with content, increases the accuracy of detection significantly as established by [17, 18].

The crowdsourcing approach uses real people to detect fake news. Platforms enable their users to flag or report posts. When a post gets enough flags, the platform can use third-party fact-checkers to verify the integrity of the post [19]. These third-parties are independent and evaluate content through evidence that is considered factual. However, [20] points out that employing these companies may have the undesired effect of dodging the responsibility for direct responsibility. Tschatschek et al. [16] Argue that the computational approach should augment the process of expert verification. Still, the flagging of posts also works as a context cue to improve the computational approach [16, 17].

B. Primary actors

Who is responsible for the detection (and handling) of fake news? Verstraete et al. [9] suggest different implementations for each of Lessig's four domains. While lawmakers can tackle the law side of the problem, and ad companies can decrease the market for fake news[20], the focus of this paper is on the architecture of the internet. The party they claim responsible in this domain is big tech, like Facebook and Google.

Detection using widgets, like NewsGuard and TrustedNews that attach to the browser put the responsibility on the user. However, [9] highlights there is little incentive for end-users to invest in solutions like these, even more so when those users belong to the most susceptible group[21]. Flagging system (in part) rely on the user. But, participation in flagging is not mandatory to benefit from the systems that use these cues if the platform implements it[16].

The researchers behind automatic detection systems mostly assume that the platform should im-

plement them [16–19, 22]. Nevertheless, the algorithms can be implemented as browser add-ins, too. Burkhardt [23] discusses a decentralized approach from companies like Factmata that platforms can choose to implement.

C. Handling fake news

When the platform is the party taking action, and that action is hiding or removing posts, then the question “Who becomes the arbiter of truth” [24]. If the company takes on this editorial responsibility, they suddenly dictate free speech and become a censoring party [20]. However, Google, Facebook and Twitter already do this as identified by [9].

A milder form of censorship is decreasing a posts visibility or the ability to spread. Facebook has already implemented this strategy on its platform [9, 24]. However, [9, 22] note the low effectivity of this strategy. Kirchner and Reuter even demonstrate that there is no significant impact on the believability of a post with decreased visibility. Even though such a post may reach fewer people, this solution fails to address the problem.

The seemingly more favourable approach is tagging posts as disinformation [9, 25]. [11] Notes that a solution must augment human judgement, instead of replacing it. Warnings attached to a piece of fake news empower users to reconsider their validity. [22] Demonstrates in a survey of 1000 participants that warning based approaches are most effective, but for better performance, they should include an explanation. A problem that warning labels fail to address is the implied truth effect [25], where multiple exposures to a statement create the illusion of truth.

D. Shortcomings

Further research suggestion and shortcomings of the examined papers serve as a rough roadmap for the fake news problem. This section will discuss these avenues.

Verstraete et al. [9] claim that natural language processing does not fare well with nuances and context (yet). However, Rubin et al. [14] refute this by training a neural network to distinguish satire from fake news by picking up on cues specific to the genre (like giving away that it is deceptive on purpose). They even propose that using deep syntax this process can be made more accurate. [16–18] Prove that the use of content cues can gain high detection accuracy. Combining these content cues with context cues allows the accuracy to get even better.

The datasets used for training and testing algorithms are often incomplete in some way. Research of fake news frequently focusses on US politics [7], possibly creating biased detection models. [17] Notes that the data they use is only in Italian. For a multi-language approach, an algorithm must train

for ground truths in multiple languages. Many researchers craft a dataset specifically for their research. Tailored datasets make comparing algorithms more complicated than necessary. With “Liar, liar pants on fire”, [26] aim to create an extensive benchmark dataset. However valuable, LIAR is still a dataset of US political news and lacking in context cues like user-flagging and network propagation.

Flagging is a very valuable context cue, combined with reinforced learning; it will likely improve detection accuracy further [18]. However, [16, 17] point out that not all users flag alike. Some users will be bad at flagging or even have malicious intent. So they propose a weighted flagging solution where user behaviour influences the cue. As of yet, it is difficult for independent researchers to implement these crowdsourced cues well, this data is almost exclusively available to the platform.

The solution to the fake news problem must take a more targeted approach to maximize efficiency. Most current suggestions on handling detected fake news work as a catch-all system. However, a small group of people is most susceptible to fake news. While the large group is relevant, the more vulnerable group should be the focus of future research according to [22]. To get more abstract models for detection, [15] states that a better understanding of the production of inaccurate information is necessary. This understanding would help discover cues for earlier detection.

E. Next steps

Overall much progress has been booked towards a solution for the fake news problem. However, there is more research to be done; the following are some conclusions on previous work and suggestions for the future:

- The first phase of the solution to the fake news problem in the architecture domain is detection. Combining the computational and crowdsourcing branches will result in the best accuracies. Further research on weighted flagging is interesting for improving the results
- With the recent significant accuracy improvements for detecting fake news using natural language processing, nuance issues become manageable.
- Most research assumes the big platforms will take responsibility for tackling the fake news problem. The solutions are frequently designed to be implemented by the big-tech.
- Research favours adding a warning to perceived fake news as a way of protecting the users. Ideally adding an explanation to the warning, a research avenue for explainable machine learning.
- Some research has proposed linking articles to a ground truth and comparing claims made but little research has been conducted on it.

Although it is impossible to establish a ground truth for everything, debunking false statements with facts is an effective counter to the problem.

- One or multiple extensive datasets with context, on different topics and languages, are still necessary for benchmarking and less biased training.
- Fake news and its environment are in a perpetual process of change, building dynamic learning models that update or learn as they evaluate are likely key to a long-term solution. User flagging and input could play an important role in model plasticity.

III. METHODS & TECHNIQUES

A. The Creative Technology Design Process

A *Design Process for Creative Technology*[27] describes a 4-phase design method for the Creative Technology Bachelor students at the University of Twente. It finds a middle ground between user-centred design and classical engineering approach. The process focusses on using existing Information and Communication Technology (ICT) with the user in mind. The four phases in the process are *Ideation*, *Specification*, *Realisation* & *Evaluation*. Figure 3 from the original paper[27] visualises the model. As can be seen, the process is non-linear and as a preferred way of working tinkering and prototyping play a vital role during the phases.

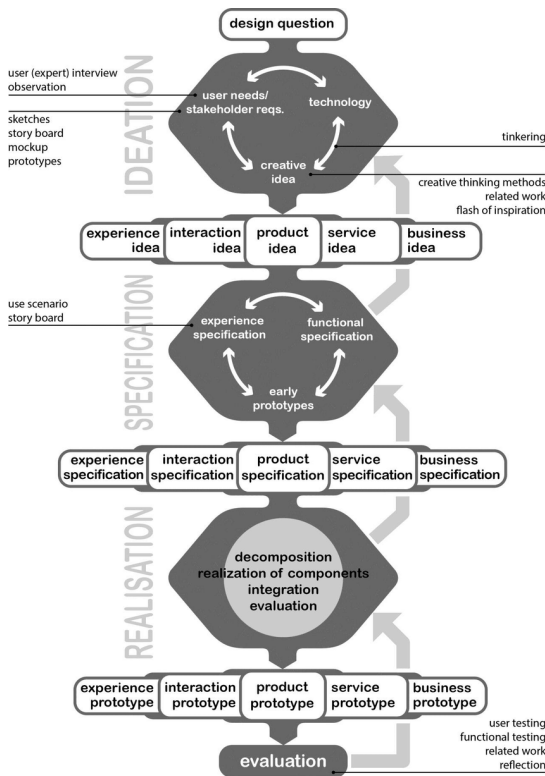


Fig. 3: Design Process for Creative Technology

The goal of the *Ideation phase* is generating and gathering ideas. Ideas can come from activities like talking to experts and stakeholders, brainstorm sessions, exploring related works or experiencing a sudden flash of inspiration.

During the *Specification phase* multiple prototypes with quick evaluation and feedback loops are used to explore the design space framed in the ideation phase. Prototypes can be improved, changed, combined or discarded. Following user-centred design, prototype testing, evaluation, and feedback create the specification for the project.

In the *Realisation phase* the specification is fulfilled. This phase is achieved by employing engineering design methods by deconstructing the specifications' demands, building those components and integrating them. Afterwards, specification requirements are evaluated.

Finally, in the *Evaluation phase* the project thus far is assessed. This step can include functional testing, verification, retracing ideation phase requirements, placing the work in the context of related work and ending on reflection.

B. Prototyping tools

1) *Language - Python*: As a language for prototyping Python 3.8 [28] is a solid choice. Python is a flexible language that is easy to write and re-write, enabling the building and sharing ideas. For the envisioned problem, rapid development and understanding code is more important than achieving the highest optimisation possible (e.g. using C). As it is a high-level programming language, there is no need to worry about memory management. Python usually runs fast enough. However, if processing times become problematic, there is always the option of using Cython [29] to speed up computations. There is also a vast online community, with tutorials, questions and answers to learn from. Another advantage of python is the many available external libraries that can be implemented quickly and relatively easily. Considering that the fake news problem is a data problem, libraries like Pandas and NumPy will be employed. The first is a tool for data analysis and manipulation and the latter for scientific computation. These are among the reasons for making python one of the most popular programming languages for data science.

2) *Environment - JupyterLab*: JupyterLab [30] is a development environment for python. This project uses JupyterLab because it offers an interactive, flexible space to write and execute code (chunks) and show visualisations in the browser. It also supports markdown and HTML for written sections between code for explanation or organisation purposes. This way, the project can be coded in chapters to show a (more or less) linear progression with the steps along the way.

3) *Version control: Github*: All of the code written and files used are publicly hosted on GitHub at the following link: <https://github.com/emielsteegh/>

Fighting_Fake_News. Next to making the work done for this project publicly available, Git allows code to be backed up and rerolled to previous checkpoints (called commits) in case something goes wrong as well as working on different features in parallel without interfering.

IV. IDEATION

In section II: State of the Art, research possibilities were identified for the wide-ranging problem statement “How can we improve the existing systems against fake news?”. This section will cover the explorative ideation phase towards a defined design space:

“Using ground truths to improve fake news detection accuracy.”

A. Divergence

Figure 2 sketches a simple process of an online fake news combatant. The two points of interest for this project are detection and handling. Improved detection leads to a higher-performing system that delivers more value to the users. Improved handling makes the action toward the user more effective. A complete fake news combatant needs all parts, but the resources are too limited for such a broad approach. By doing divergent research and building a good understanding of the context, one of the two blocks can be selected as a focal point for this project.

1) *Related works*: Section II has covered some related works from the scientific community, but what about the employed products and services? FactCheck.org, Politifact.com, Snopes.com and IFCN are among the most popular players of the anti-fake news game. The following section is a review of these services.

FactCheck.org is “a nonpartisan, nonprofit consumer advocate for voters that aims to reduce the level of deception and confusion in U.S. politics.” [31] They focus on fact-checking known public figures and entities. Their approach is manual through research teams. They monitor news outlets and political events and write thorough articles to verify or debunk claims. These articles are available on their website.

“PolitiFact is a fact-checking website that rates the accuracy of claims by elected officials and others on its Truth-O-Meter.” [32]. Politifact is also heavily U.S. politics-oriented. They also have a system to track whether officials follow up on the promises they make. Their process is a manual one, like FactCheck.org done by a small team of researchers using sources to check statements hosted on their website.

Snopes [33] is a larger company than the previous ones. Unlike the previous two, they concern themselves with anything submitted by enough of their members, including U.S. politics among many other

topics. These submissions come in many forms like articles, images, videos and messages. However, their process of reviewing the submissions is similar; a team of researchers uses the unbiased sources available to come to a veracity rating and explanation. All the work they publish is available on the Snopes website.

The IFCN (International Fact-Checking Network) [34] supports existing fact-checking services through funding and providing a code of principles, among others. They provide media literacy training to individuals and host a list of “verified signatories”: fact-checking websites and businesses that adhere to the IFCN code of principles. All three websites above are on this list. Being on this list signifies that a company is honest, reliable and transparent. However, the IFCN provides no fact-checking service of their own.

2) *Stakeholders*: As an exercise to generate ideas and get a better grip on the context, I identifies the various stakeholders of systems against fake news.

The intended end-user groups of such a system can be split in two: *group a*, the people that consume content on the internet who are vulnerable to fake news because of possible biases, emotional or lacking media literacy targeting [35]. *Group b*, on the other hand, are advocates of a fake news combatant and are actively looking to protect themselves; these are the people that may already browse websites like Snopes. Both *group a* and *b* behave vastly different, and both have to be considered during the design process.

Group c consists of the outlets where fake news can exist. These platforms include social media, blogs, news websites and search engines. Actively and openly fighting fake news can create the impression of higher trustworthiness than a platform that hosts misleading content. On the other hand, fake news spreads faster and wider than real news [3], and more reach leads to more user attention. Furthermore, attention is where a website that hosts content can create monetary value. So one could argue that fake news can create more revenue.

Group d consists of professional and amateur journalists who create actual news content. According to the definition adopted at the start of this report, fake news must be in a journalistic format, the people who create genuine news are the journalists. They stand to benefit from a fake news combatant as it can help preserve the journalistic format’s integrity.

Group e are the people that generate fake news. A system against fake news potentially harms these people as their lively hood can depend on the money they earn by creating it. They probably have little positive interest in the system but may actively resist it.

3) *A blockchain approach*: An interview was conducted with WordProof[36] to diverge the research space further. WordProof is a company with a novel approach for dealing with fake news on the web. The cornerstone of their vision is transparency and accountability. Using blockchain technology, they times-

TABLE I: Stakeholders of a fake news combatant

	Stakeholder group	Description
<i>a</i>	Vulnerable readers	People who are (likely to become) fake news victims
<i>b</i>	Advocate readers	Want a tool or system to protect themselves from fake news.
<i>c</i>	Media platforms	The websites that host news(like) content
<i>d</i>	News publishers	Journalists and writers, the people that create the (real news) content
<i>e</i>	Fake news abusers	People that (deliberately) create and spread fake news.

tamp web pages to build trust and protect content. Their focus lies on creating reliability online without becoming internet police. They want to play an enabling role. Their idea for a solution consists of a blockchain identity network based on the World Wide Web Consortium (W3C) Decentralised Identifier and Self-Sovereign Identity. In this network, the identity is not renewable, verifiable, and transparent. How far a post or article can spread is based on how much of their identity the creator wants to reveal. Full transparency of the creator grants full access to reach; remaining anonymous limits reach to friends only. This approach could create a self-regulating reputation market where spreading fake news has negative social consequences for the creator.

B. Convergence

The current ideation space is rather extensive, to successfully move to the specification phase; decisions must be made. Firstly, what block of figure 2 should be the focus of this research project? Both are exciting and viable ventures. However, during all of the ideation phase, the detection has sparked more interest. So after a period of deliberation, the decision fell on detection.

There are advantages that this project envisages in automatic fake news detection. It must be noted that automation does not replace the work humans do for a sound functioning system. Automated detection of fake news should exist to support researchers. Perhaps detection can be sped up with machine assistance, but eventually, a decision needs a grounded explanation. Most of the fake news combatants share all of their work on their website. Automated detection can help as a step to take an article debunking or verifying a statement to its source. A potential use case is: A user finds a piece of potentially fake news online; the automatic detection system kicks in to give an early assessment. Then it sends a notification to the research team for a new review or lets the user know of an existing article addressing the veracity.

Automatic fake news detection already exists, as discussed in section II. Most of the current works focus on models using patterns inside the text, or its context to reach a verdict. This research project will work towards improving detection accuracy using some novel approach. One such novel approach is mimicking the one research teams use; they find facts to compare to a potential piece of fake news. In existing research, this seems like a little investigated niche. So the next section will explore and specify *how*

ground truth can be used to improve fake news detection accuracy.

V. SPECIFICATION

This section will set early requirements in order to guide the modelling process. [37] is one of the few papers discussing a ground truth model; “a systematic approach for automatic fact-checking of news has to the best of our knowledge never been presented before”. However, they do provide a theoretical model for fact-checking. Figure 4 is a adaptation of their model. The letter K in the figure means *knowledge*.

An advantage of the ground truth approach is that the knowledge base can be extended and updated as more information becomes available. This makes it a scalable approach. If a new topic comes up and information exists about it, it needs to be extracted and added to the knowledge base. If everything works, the latest information is taken into account for all new decisions.

A. Early requirements

The model that comes out of this project is specified around figure 4. The core concept comes down to two phases. To fact-check news automatically, there must exist a knowledge base or ground truth to use to compare a new piece of information. The first phase uses a processing method to obtain this knowledge base from the information publicly available. The second phase uses similar processing steps to extract the news article’s knowledge and compare it to the knowledge base. From that comparison, the model reaches a decision about the veracity. This description creates the *requirements 1, 2, 3 & 4* in table II.

The model should also perform well enough. Two metrics to rate performance are accuracy and time. The accuracy metric should be better than a ZeroR algorithm; always choosing the most frequently occurring option. If a dataset consists of 55% real news and 45% fake news, the model should do better than 55%. This seems like a meagre objective, and it is. The goal is not to build the perfect model but to be able to improve existing ones. The theory is that adding more information (like a comparison to ground truth) to an existing model will improve it. So, the accuracy metric sets a precedent for *requirements 5 & 6* in table II.

The final metric time is vital for usability. If the process of detection is too slow, the project is

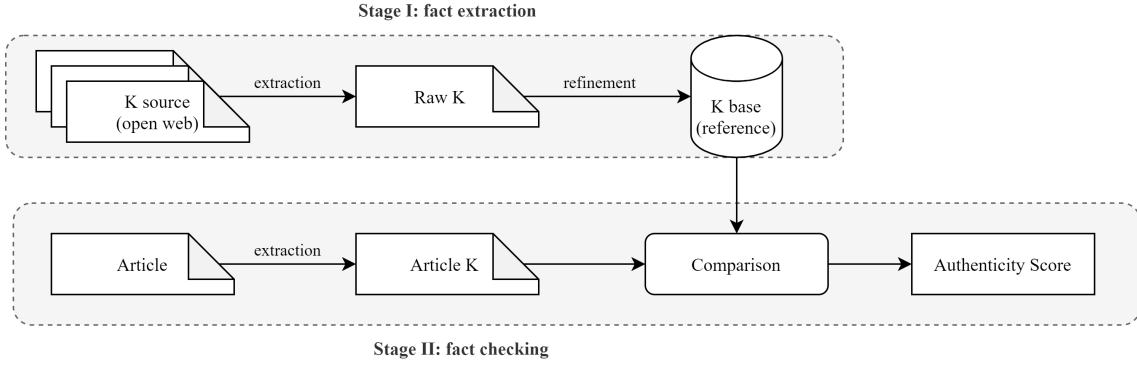


Fig. 4: Knowledge extraction and comparison

severely crippled. Say a user is reading an article and only halfway through something happens, that could mean half of the information is consumed with the belief that it was truthful. If a team of researchers were to employ an automatic detection tool for preliminary filtering of fake and real news, but it takes a day because there are thousands of articles, they would likely not want to use it. Being timely is an essential concept in the news, a tool that hinders the workflow more than it aids will probably not be used. This creates the final *requirement 7* in table II.

B. A representation of truth

What knowledge means has to be specified. The raw form where the knowledge will come from is a text written by humans (or computers). Two interpretations of knowledge representation were selected to work out for this project: RDF triples and sentiment.

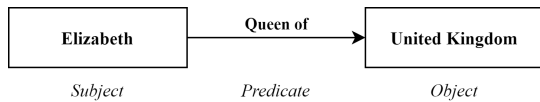


Fig. 5: Example of an SPO

1) *Semantic triples*: A semantic triple allows for text to become machine-readable. A triple exists of

three parts: the subject, the predicate and the object (SPO); the subject performs an action, the predicate, on an object. Figure 5 is an example of such a relation derived from a sentence: “Elizabeth is the queen of the United Kingdom.”

A triple can be extended into a network, where a subject is also an object in another triple. A triple can even be a subject or object in itself in cases like: [Bobby - says - [Elizabeth - queen of - U.K.]]. Sentences can contain multiple triples as they become more complex. Rules like these allow the building of a knowledge base in the shape of an extensive triple network [38].

A well-developed language for triples is the Resource Description Framework (RDF) in the Extensible Markup Language (XML) [39] and is common in relational information storage on the web. Wikipedia, for example, is full of RDF/XML triples. These triples should be usable in the construction of a knowledge base or ground truth, and the representation of new articles [40]. The comparison then becomes a graph or network theory problem.

2) *Sentiment*: The sentiment of a sentence is a representation of how positively or negatively it is written. In this way, it becomes possible to tell how an author feels toward a particular topic or even tell if they agree or disagree [41]. Sentiment analysis is a natural language processing method to assess this emotion in a piece of data like a sentence. The analysis may be based on machine learned patterns, rule sets or a combination of the two [42]. Influence can include the positive or negative connotations

TABLE II: Model requirements

Early requirements	
1	The model can extract knowledge from raw information
2	The model has a knowledge base
3	The model can compare a new knowledge to the knowledge base
4	The model can conclude a comparison
5	The model should be more accurate than the baseline
6	The model can improve an existing model
7	The model can process information in a timely manner

TABLE III: Reviewed available datasets

Name & Link	Set size	Veracity label	Topic(s)	Notes
McIntire https://github.com/lutzhamel/fake-news/tree/master/data	~8000	Boolean	U.S. Pol.	popular in other research
Liar, liar pant on fire[26] https://github.com/thiagorainmaker77/liar_dataset	~12800	Likert (6)	U.S. Pol.	single statements, popular
FNC https://github.com/FakeNewsChallenge	~5000	-	World news	title-body stance detection
FakeNewsNet https://www.kaggle.com/mdepak/fakenewsnet	~400	Boolean	U.S. Pol.	includes context
Kaggle B https://www.kaggle.com/c/fake-news/data	~20800	Boolean	World news	multi-lingual
Kaggle C https://www.kaggle.com/mrisdal/fake-news/data	~13000	Only fake	World news	fake news only, multi-lingual
ISOT Fake News https://www.uvic.ca/engineering/ece/isot/datasets/	~44900	Boolean	U.S. Pol. & World news	popular in other research
Snopes Checked https://github.com/sfu-discourse-lab/MisInfoText	~300	Likert (5)	World news	includes some context

words carry, emoji use, grammar or deeper syntaxes and many more features. The sentiment is usually represented as a scale from -1 to 1; negative to positive with neutral in the middle (0). Constructing a knowledge base using sentiment data could prove to be a valuable comparator for unverified articles.

C. The dataset

A final specification that needs to be made is the dataset to use. There are some requirements that the set has to fulfil, namely *the dataset must contain*:

- 1) the text of the articles
- 2) real news and fake news
- 3) an accurate veracity label per article
- 4) a large enough collection of articles

The cosen definition for fake news means that the dataset should contain (fake)news articles. A veracity assessment is necessary for training the model, building a ground truth and verification of the model. Since *requirement 2* requires building or obtaining a knowledge base to compare things to, the dataset must be extensive enough. A dataset that is too small will create a ground truth with more holes than information, likely resulting in inferior results. Then there are some datasets qualities that, if the choices allow it, would be *nice to have*:

- 5) popularity among other papers
- 6) an extremely large collection of data
- 7) diverse topics
- 8) diverse sources
- 9) contextual data

The benefits of these additional requirements could improve accuracy or be valuable in other ways. If a dataset is popular, it can be used as a benchmark, comparing this project's achievements to another work. If the dataset is extensive ($\geq 20k$ entries) processing times will increase, which is a minor trade for a lower estimation variance and better prediction [43]. However, not all data is valuable, especially in a large set; to keep the data relevant individual samples must be examined. A broader range of topics decreases the chance of overfitting. If the ground truth only covers a single topic (like the United States presidential elections) the model will likely generalise very poorly to other topics. A diverse set of sources helps prevent biases that may come with having a single website as a source. Examples of these biases are a dependency on a certain writing style or angle on the subject. Finally, although contextual data is not envisioned in the current specifications or design process, it allows the model to be combined with different models that *do* use contextual data.

With the information obtained in this section, a choice for a dataset can be made. There are many datasets readily available online, published by various organisations, for different reasons. Table III was created as a compilation of some of the more commonly used datasets that can be found online to make a well-founded choice. (Note to the reader: there is a section where the Snopes Checked set is used to try something different). The choice fell on the ISOT dataset.

The ISOT Fake News dataset X Ahmed 2018s [] has a large volume of data and slightly more diverse topics than most other available datasets. The set consists of two separate files: True.csv (21417 'real' news articles) and Fake.csv (23481 fake news articles)

for a total of 44898 articles in a 48/52 split between real/fake. The files contain the articles’ text body along with the title, category, and publishing date. The articles are mostly from the years 2016 and 2017. The true category articles are scraped from reliable news sources, while the fake ones come from verified fake sources. The texts are processed to get rid of textual hindrances like ads and HTML code.

VI. REALISATION

Based on figure 4 and the specification defined in section V there is enough information to start working on the prototypes. From those early models, incremental improvements can be made using sample testing, evaluation and literature. The setup uses separate Jupyter Notebook files for each “large” step. These files are called labs; each sets out to achieve a goal, and when that goal is reached, the code will be made more compact and used in the following lab for a different goal.

A. Lab 01 – Sentiment analysis

The goal of the very first lab is to get the sentiment analysis running. There are many ways to achieve sentiment analysis; build a custom model or use one of the publicly available ones to python. Since sentiment analysis is not a goal, but a tool in this project, choosing an existing one makes the most sense. The python Natural Language Processing Toolkit (NLTK) is the most popular (and probably extensive) python package for Natural Language Processing (NLP) available. [44] is a well-structured tutorial showing some of the different sentiment analysis options that NLTK offers, how they were created and how they work.

TABLE IV: sources of sentiment tools

technique	word source
NLTK (VADER)	everywhere
TextBlob	product reviews
TextBlob + NaiveBayesAnalyzer	movie reviews

After following the tutorial, there were three working techniques, ready to judge sentiments: VADER, TextBlob & TextBlob+NaiveBayesAnalyzer. Table IV outlines the sources used to train the analysis models. As the original paper of VADER [45] describes, models trained on specific topics or sources may outperform a more general model in their own domain. However, they will generalise worse to new topics and sources. Considering that this project is supposed to work across various topics, choosing VADER makes the most sense. So for the coming labs, VADER will be employed to derive sentiments from statements.

VADER or Valence Aware Dictionary for Sentiment Reasoning works by summing up each word’s sentiment values. By letting many humans assign

sentiment values to many words, the creators have formulated a dictionary that rather accurately reflects people’s feelings towards words. However, language is not so simple as a dictionary. Some words act like modifiers; adverbs like “not” invert meanings and those like “very” and “slightly” change the intensity of other words. Capitalisation can also change the intensity of message. VADER picks up on these modifications, table V illustrates a simple example. When a sentence is processed using VADER, it returns four values: the degree of positivity, negativity, neutrality (in the range [0,1]) and a compound score combining the previous three (in the range [-1,1]). This project will use the compound score, some nuance may be lost, but the score is accurate enough. This lab’s code can be found in appendix A or on the project GitHub.

TABLE V: VADER applied to modified sentences

Sentence	Sentiment
I am <i>not</i> happy	-0.4585
I am <i>slightly</i> happy	0.5279
I am happy	0.5719
I AM HAPPY	0.5719
I am <i>very</i> happy	0.6115

B. Lab 02 – Triple extraction

The goal of the second lab is to enable triple extraction. The only option that came up to achieve this task was the Stanford OpenIE[46], a Java implementation of [47] in a python wrapper. It is part of Stanford CoreNLP, and takes quite some effort to get working package wise. But once it works, it proves to be a powerful tool. In the lab, the following text:https://github.com/emielsteegh/Fighting_Fake_News/blob/master/labs/sources/article_corona_vaccine.txt

is transformed into a triple network. Using the GraphViz package, which also has a more complicated install process than most packages, the triple network is visualised as follows:https://github.com/emielsteegh/Fighting_Fake_News/blob/master/labs/out/graph.svg.

(Note: The image was far too large to be placed in the appendix)

The Stanford OpenIE implementation works by splitting each sentence into clauses. These clauses are shortened as much as possible resulting in sentence fragments. Those sentences are then turned into OpenIE triplets using natural logic [46]. These triples are then stored for the user to employ.

As a next step building a knowledge base is not necessary for a model that uses semantic triples. Wikipedia is the most extensive collection of free knowledge. A large part of this information is available as a knowledge graph under the DBpedia project [48]. This information network is accessible through SPARQL. The extracted knowledge from the article in the previous paragraph could be compared to this network. Adapting figure 4 to the the triple approach

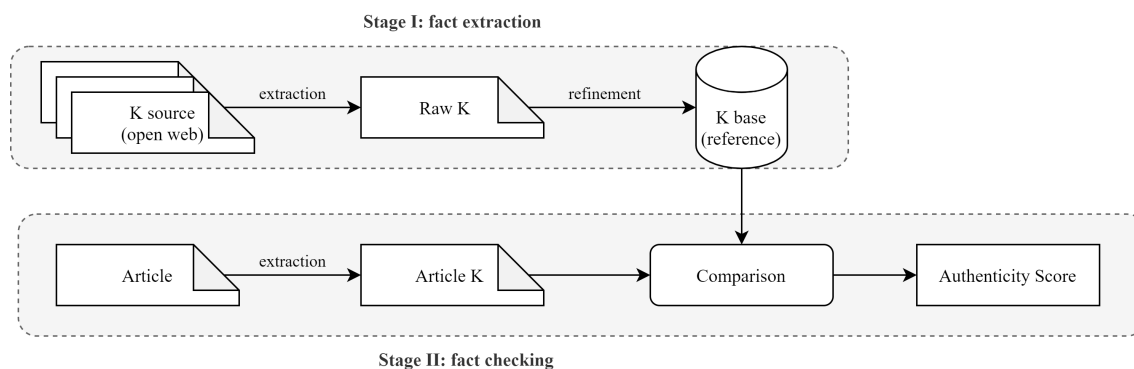


Fig. 6: Triple extraction and comparison model

with DBpedia results in figure 6. However, there are some significant problems:

- 1) Knowledge gaps.
Consider the triple (Jack;president;moon). The “moon” object most likely has no president, and the predicate “jack” could be any person. A knowledge graph like DBpedia can only be used to verify true statements, while uncovered knowledge does not make it false. The triple (house;painted;yellow) may very well be true, but such a statement cannot be verified as long as the relationship does not exist in the graph.
- 2) Speed.
Generating triples from a medium-sized article like the one used in this lab takes half a minute on a decently powerful machine. It resulted in a little over 400 triples. Then they have to be compared to over 4.5 million entries in the DBpedia knowledge base.
- 3) Accuracy.
Although the results of OpenIE are impressive, for larger bodies of text, they are also too messy. The over 400 triples contain many duplicates and unnecessary information.

The first problem can likely be solved or worked around. However, to attain a workable speed, this project will need optimisation or a form of cloud computing outside. Solving the third problem would need a form of complex preprocessing. These latter problems are most likely beyond this project’s expertise and scope. For these reasons, RDF triples are abandoned as a form of knowledge that is achievable. The project will continue to focus solely on using sentiment as a knowledge representation in the following labs. This lab’s code can be found in appendix B or on the project GitHub.

C. Lab 03 – Sentiment model

Now that the decision has been made to pursue sentiment as knowledge, it is time to work out how sentiment should be employed. This lab develops the idea of a sentiment sequence for each article, a series of sentiment values, one for each sentence in order.

A custom module (ArticleModule.py on the GitHub) was written to store articles along with their sentiments and statistical features. It also included the definitions for processing functions. However, this support module was abandoned in favour of using the Pandas Dataframe to store data. They make the process easier, faster and more scalable. Dataframes include operations for viewing and manipulation, and the NumPy integration enables faster calculations. The following lab will deal with turning everything so far into a smooth pandas experience.

From this lab on the NLTK Punkt module is used to split articles into sentences. A manual rule-based sentence splitter function was adapted, too. However, Punkt was faster and outperformed it consistently. These split sentences can then be processed one by one using the first lab’s VADER approach to reach a fingerprint. To generate more data NumPy’s describe is used on the fingerprint. Describe delivers statistical features of a set of numbers: the count, mean, standard deviation, min, interquartile values and max. This description creates some more complex data that may prove to have value in decision making at a later stage.

As a final step in this lab, some fingerprints are visualised to understand the data better. The dataset is loaded in, and a sample of four authentic articles are compared with four pieces of fake news. Using Matplotlib figure 7 was created. Figure 7 does give any clear conclusions, but after viewing many of these graphs, some observations can be made. Articles vary in length greatly, going from 5 or fewer sentences to over 30. Varying length is likely something that needs to be dealt with in a later lab. Articles tend to centre around neutrality; it cannot be said that fake news or real news tends more towards positivity or negativity than the other. Both seem to be slightly more positive than negative. This lab’s code can be found in appendix C and D or on the project GitHub.

D. Lab 04 – Sentiment pre-processing

The goal of this lab is to process the entire dataset to sentiment sequences. In the previous lab, individual articles were processed into their fingerprints. That process must be scaled up to deal with the

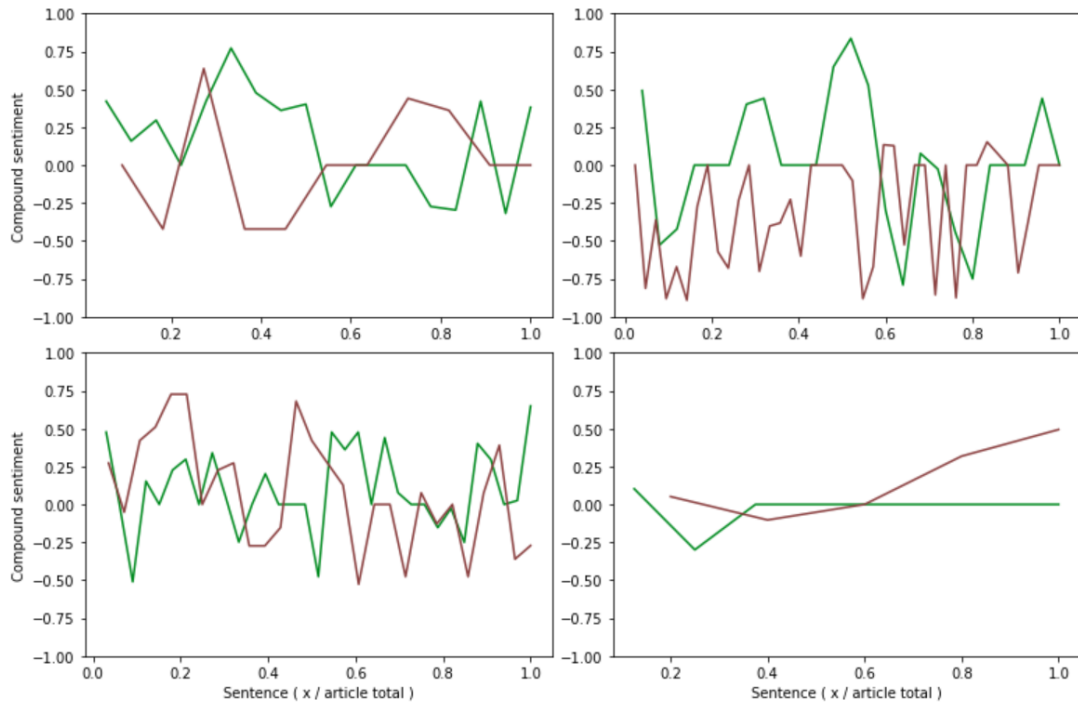


Fig. 7: Sentiment sequences of 4 fake (red) and 4 real (green) articles

40.000+ instances of the ISOT dataset. Optimisation is essential here. The VADER module is fast, and the NLTK Punkt module is fast too. However, if the total time per article averages around 200ms, the entire processing will take around three hours. So using for-loops to go over the set is not an option.

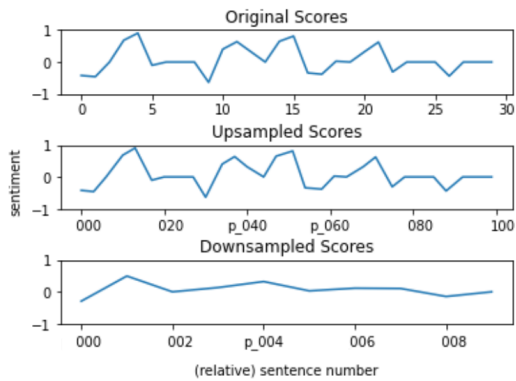


Fig. 8: Up- and downsampling a sequence (from 30 to 100 and 10)

The preprocessing starts with loading the dataset, then describing the individual functions. The functions required are: splitting text into sentences, sentences into sentiments, a set of sentiments into statistical features, and then resampling the sentiment sequence so that everything is of equal length. There

are a few ways to go about this, resampling (up or down) or clipping and padding. The two different sampling methods are shown in figure 8. In upsampling (or interpolation) data is created between the points to fill the space, e.g. allowing twenty points of data to become one hundred. In downsampling, the data is compressed to fewer points by taking the mean of buckets of data. In the case of padding and clipping sequences that are too short are filled with a padding number that is to be ignored later. Sequences that are too long are cut off to the amount of desired numbers, either removing them at the start or end of the sequence.

By performing some simple data analysis on the sentence count of all articles, it becomes clear that more than 99.5% of the articles contain under one hundred sentences. Downsampling to a too low number will remove nuance but has the benefit of being easier to process. Upsampling does the opposite, choosing a number that encapsulates all possible sentence counts preserves all nuance but creates an unnecessary amount of data, making using the new set more difficult. Clipping and cutting is fast and does not create data that does not exist but seemed like a too lossy approach in this stage, so sampling was chosen instead. A sample rate of 200 sentiment points per article was selected to accommodate for longer articles.

With all the functions ready, the dataset is shrunk down to 8 instances to avoid getting stuck waiting for the entire corpus to process in the likely case that it does not work the first time. Applying the code to such a small batch allows for quicker testing and de-

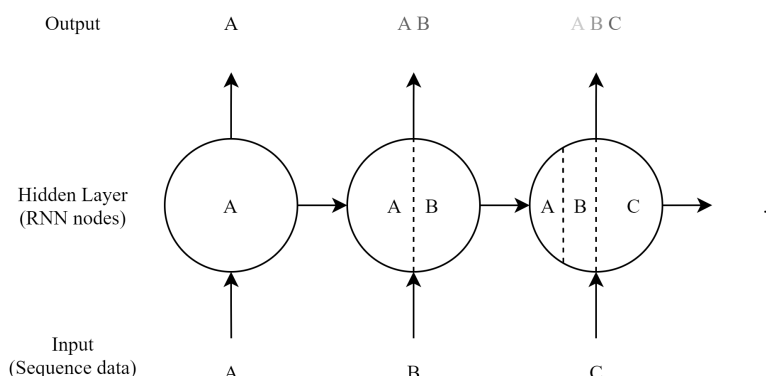


Fig. 9: Simple illustration of the vanishing gradient problem

bugging. All the functions were turned into a single efficient function use on the article text column of the DataFrame through Panda's *apply* function. *Apply* is a much faster way of modifying a DataFrame than looping over the data individually because it makes it a vector function. Finally, when everything seemed to be in working order, the process was started for the entire dataset—resulting in a processed dataset containing the veracity label, statistical features and the sentiment sequence spread over 200 columns. The whole process took only 6 minutes and 23 seconds, so it seems like the optimisation decision paid off. Finally, this set is exported as a .csv file for use in Weka and following labs. This lab's code can be found in appendix E or on the project GitHub.

With the dataset exported, it can be used to obtain the very first results. Weka [49] is an open-source machine learning software that can be used for powerful data mining and analysis. First, the dataset is loaded into Weka and the veracity label is selected as the class to identify. Using the ZeroR classifier gets the baseline of $\sim 51.6208\%$. ZeroR chooses the class that occurs most frequently for every instance. Since there are more fake articles than true ones, it predicts fake every time and does slightly better than random. Running the Weka default implementation of multinomial logistic regression on the entire dataset results in a classification accuracy of $\sim 58.38\%$. However, when going back one step and removing the sentiment sequences from the dataset, thus using only the sentiment statistics, the accuracy rises to $\sim 59.65\%$. This means that in the current approach the sentiment sequences only cloud the machine's judgement. Still, sentiment statistics offers a little over 6% increase from the baseline. A promising sign that there may be better results to be obtained with better modelling.

E. Lab 05 – Recurrent neural network

The sentiment sequence approach needs to be revised. Lab 04 showed promise in the overall sentiment approach, but the realisation of the sequences was not adequate. Since the problem is in sequence form, using logistic regression makes little sense, the patterns the model is looking for do not have to exist

in the same location. Article sentiments are ordered sequence data, and the order may very well matter. This is where RNNs (Recurrent Neural Networks) [50] come in. Recurrent neural networks are neural networks (like logistic regression used in the previous lab), but they have closed-loop feedback systems. This feedback allows them to “remember” what happened in a previous part of the sequence. RNNs “have addressed problems involving dynamical systems with time sequences of events” [50] much like the system at hand.

The basic RNN tends to forget things or remember them too well for long sequences of data. This behaviour is called the exploding and vanishing gradient problem [51, 52]. The nodes in a default RNN propagate their value to the next node. The next node gets the next sequence value as well as the value propagated by the previous node. If the weight of the previous node's value is smaller than or equal to the new sequence input, the value will vanish. Figure 9 was created to illustrate this problem better, with a 50/50 split between sequence and previous node input, A only counts for 25% of the weight. The opposite happens when the previous node is weighted stronger than the new sequence input. In that case, the exploding gradient problem happens, and A will overpower other sequence values. With shorter sequences or ones where only the first or last part is significant, this matters less. However, the sequence data coming from the articles are long and distributed. So, a solution is necessary.

The LSTM (Long Short-Term Memory) is a type of RNN that seeks to remedy the vanishing/exploding gradient problem [53]. The LSTM node has a more complex architecture than the default RNN node. It takes the previous node's input and the sequence input; it takes a memory state input and creates one. Inside there are gates that allow the nodes to “forget” and “remember”, making the model much better suited for longer sequences and more complex data.

The TensorFlow module for python offers access to the Keras deep learning API. Keras can be used to build and implement LSTM models. However, the layers take NumPy vectors as inputs, so lab 4 has to be reworked to fit this input requirement. Since the

sentiment pre-processing needs changes and Keras allows masking layers, clipping and padding makes more sense than sampling. Using a padding value like -10 that does not exist in the sequence data does not interfere with the data. No unnecessary manipulation like the creation or degradation of data has to happen, which is likely better for accuracy. The masking layer will let the model know to ignore the -10 values before taking the sequence. The sequence length is determined to be capped at 100, longer sequences mean longer processing and building times, and anything longer will have to do with only the last 100 sentences.

With the data prepared once again, it is time to shuffle it and split it into a training and testing set of inputs (X) and outputs (y). The test set comprises about 40% of the total data, a large enough chunk for evaluation. Both the training and testing sets have about the same division of classes to reduce biases.

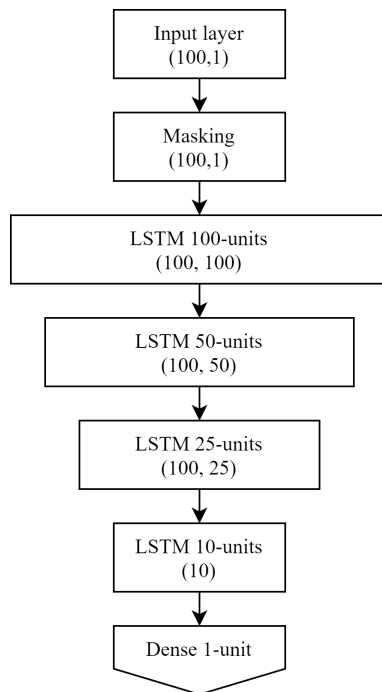


Fig. 10: Final LSTM-model design

The design of the stacked LSTM was a bit of trial and error. More complex models take much longer to train than simpler models, and more complexity does not keep adding value forever. A too simple model will be unable to make sense of the data. The model design constraints require it to begin with an input layer, then a masking layer and end on a dense layer. The dense layer with a sigmoid activation function at the end will flatten whatever comes before it and change it to a value in the range [0,1] from fake to real. An essential addition to the model is the use of dropout layers in between. Dropout layers set random input units to zero. Doing this prevents overfitting. Without these layers, the LSTMs (especially in a stacked design) will learn the entire dataset achieving 100% accuracy on the training set

and scoring near 0% on the test set.

The middle part of the model, between the masking and the dense layer, are open for trial. The start was a simple LSTM layer of 100 units, matching the maximum input data, getting accuracies of around 60%. More experimenting led to deeper (stacked) LSTMs with smaller layers, the deeper they get performing better. Finally, the design settled on a deep narrowing model described in figure 10. After 40 epochs of training with a batch size of 64, the model achieved an accuracy of 64.1%. This lab's code and the training history (loss and accuracy) graphs can be found in appendix F or on the project GitHub.

F. Lab 06 - Snopes Improvements

This lab was meant to improve the LSTM and use it on the Snopes dataset but the attempt was abandoned. The architecture improvements were moved to lab 5. However, the code is still available in appendix G and on the project GitHub.

G. Lab 07 – Sentiment & Topic

Using a sentiment sequence is hardly a ground truth approach. Yes, any new instance is ultimately compared to what is known as true and fake. However, the ground truth theory focusses more on whether a statement is true. Using sentiment to figure out whether true articles agree or disagree with a statement or topic was the proposed idea. To achieve this, it is necessary to pair the article's topic to the sentiment and the veracity label.

Ideally, this lab would result in a sentence level topic model. However, that is far beyond this project's scope, considering that sentence-level topic detection is already extremely challenging with the current state of the art technology. A related approach that is achievable is article-level topic modelling using LDA (Latent Dirichlet Allocation) [54]. LDA is an NLP machine learning model that can automatically discover topics across a corpus of documents. It views each document in the corpus as a mixture of various corpus-wide topics. Each topic is observed as a distribution of words, and each document consists of a distribution of words. How well the document distribution fits the various topic distributions gives away which topics it is likely to belong to. The Gensim module for python is the most popular topic modelling library; it is easy to work with and has solid documentation.

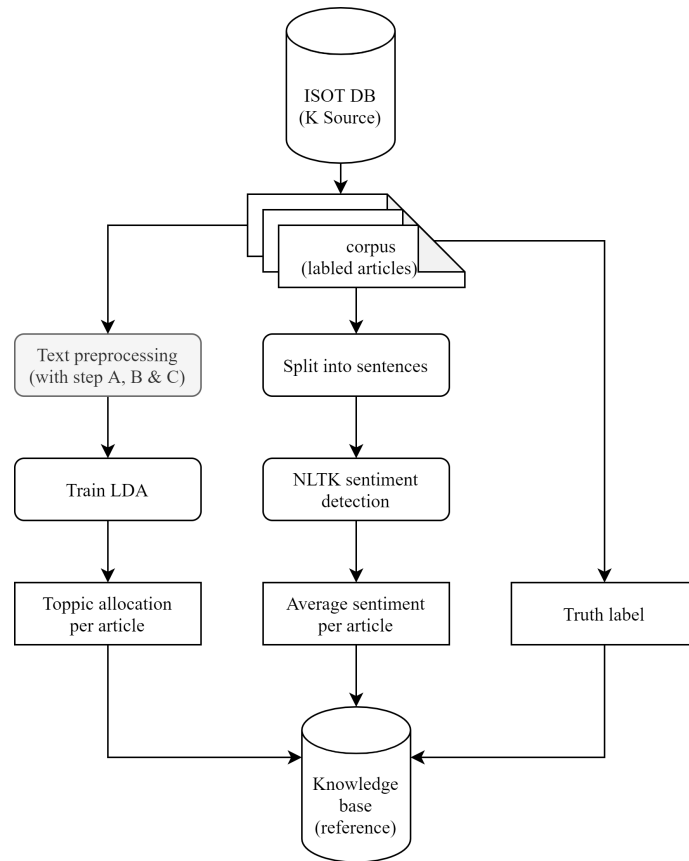


Fig. 11: Sentiment & topic knowledge base creation architecture

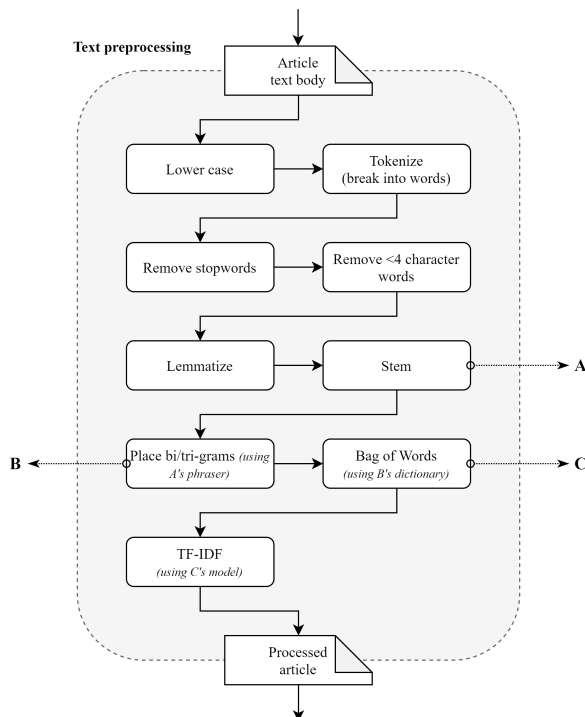


Fig. 12: Text preprocessing method

1) *Getting started*: Figure 11 outlines how to build the sentiment and topic knowledge base. The first task in getting to the topic allocation is pre-processing the entire text. Because this is a long process, it is outlined further in figure 12. The goal of pre-processing the text is to get it to a machine-readable format, a numeric vector. Pre-processing start with removing all the capitalisation form the text, the meaning of "Election" is the same as "election". The next step is tokenising, using Gensim's simple pre-process function, a document is split into individual words. If the word is removed if it is on the list of stopwords: supporting words without topical meaning (e.g. *it, the, than, but*). The word is also removed if it is three characters or less, as they likely contribute little meaning.

2) *Lemmatisation and stemming*: Next is lemmatisation, changing words to their base form (e.g. *"better": "good", "are": "be", "playing": "play"*). Lemmatisation uses a dictionary of words and their roots to transform them. In the following step, the words are stemmed. Stemming is similar to lemmatisation but works on a rule based method removing common prefixes and suffixes (e.g. *"troubled": "troubl", "caption": "capt"*). This makes stemming less accurate than lemmatisation. However, the advantage of stemming is that it is fast and includes any words the lemmatiser missed because the word did not exist in the dictionary. The specific stemming ruleset used is

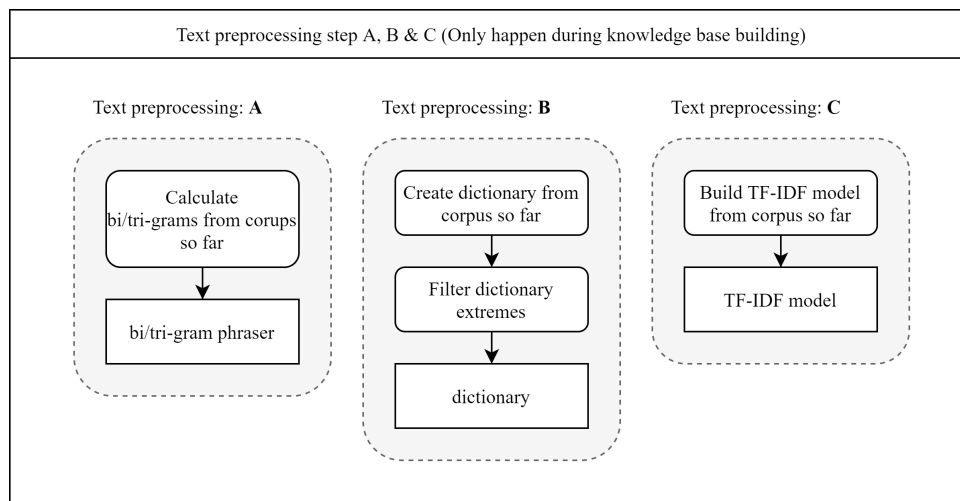


Fig. 13: Additional steps during first time text preprocessing

Porter and the lemmatiser comes from *WordNet*.

3) *N-grams*: The next step is placing bi and tri-grams in the text. A bigram is a pair of words that commonly occurs together like "white house", it makes sense to change them into one word for the machine "white_house" as this combined word has a separate meaning from the words apart. A trigram is the same thing with three words, like "president_barack_obama". In order to figure out which word combinations are a good fit for these n-grams, the entire corpus must be processed so far and fed to Gensim's Phrases module. This module figures out which words frequently occur together relative to their total separate occurrences. If the combination exceeds a certain threshold, they are added to the bi or trigram dictionary. Choosing the right threshold values determines whether the n-grams will add value or create chaos. After some experimentation the threshold settled on 20, the min_count remained at its default value 5. The n-gram calculation process is step A in figure 12 and 13. Step A only needs to happen once: during the initial processing of the corpus.

4) *Dictionary and bag of words*: With the n-grams placed, it is now time to turn the text into a bag of words. A bag of words is a text representation that assigns words an ID based on their occurrence in the model's dictionary and counts how frequently that word occurs. In order to transform a document into a bag of words, a dictionary is needed first. Creating a dictionary is a side step (B in figure 12 and 13) that needs to be performed once during the initial processing of the corpus. The dictionary first consists of all the words in the processed corpus so far, in alphabetic order, for the corpus so far that are 116,439 words. The dictionary contains far too many words, most likely have little influence on the topic. So, extremes are removed; words that exist too frequently and too infrequently and the dictionary is down to 21,592 words. Using this dictionary, each document is turned into a bag of words. If a word in the sentence is not in the dictionary, it is ignored.

5) *TF-IDF*: The bag of words is turned into a TF-IDF (term frequency-inverse document frequency) representation. It is a statistical measure of word frequency related to the frequency of the total occurrences. To turn the bag of words into a TF-IDF one last side-step must be made during the first processing (C in figure 12 and 13). The Gensim TF-IDF model function takes all the bag of words made from the corpus and turns it into the new model. With every document in the corpus turned into TF-IDF representation, pre-processing is done, and the LDA can be trained.

6) *Training the LDA*: Using Gensim, training the LDA is only a matter of plugging the pre-processed corpus into the LDA Multicore function. The multicore variant allows the machine to calculate the model using all except one core, considerably speeding up the process. The most important action during this stage is choosing the "best" parameters for the model. The evaluation metric to check if everything went well is the coherence score, calculated by Gensim's coherence model. The score range from 0 (no coherence) to 1 (perfect coherence). If the score reaches one, something went wrong; perfect coherence should be impossible to achieve in a corpus like this one. It seems that 0.6 is a good goal for the score. The number of topics is the essential parameter to set. It signifies how many topics the LDA is looking for—picking too few results in large, abstract categories—choosing a too high number results in micro categories that are unlikely to match new data or make sense in the real world. Using the coherence model and the PyLDAvis module to view and explore the modelled topics (some examples in appendix I), a total of 15 topics—with TFIDF input instead of a bag of words—was selected that achieved a coherence score of 0.53. This lab's code can be found in appendix H or on the project GitHub.

7) *Results*: To check whether topic modelling was effective to some degree the topic predictions of each article are combined with their sentiment average. This data is exported to a csv file again and loaded

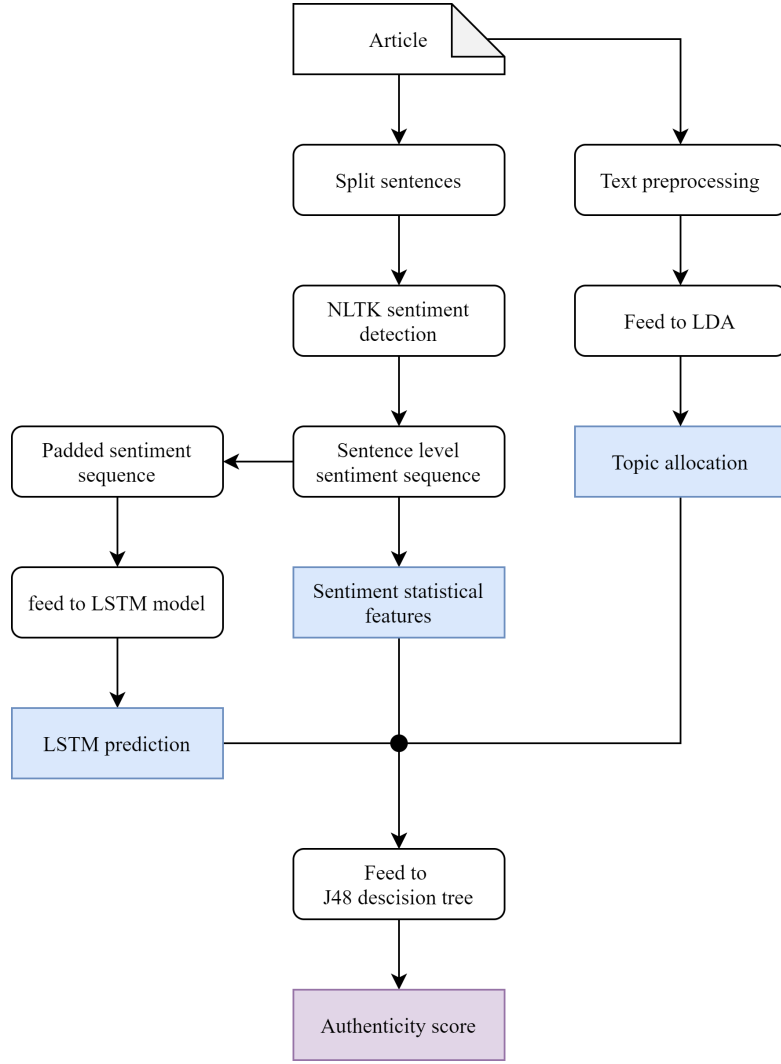


Fig. 14: Final knowledge base creation model

into the Weka explorer. The J48 model at the default weka implementation with a modification of a minimum of 50 leaves (to prevent overfitting) per branch resulted in the accuracies displayed in table VI. From this table the observation can be made that although the average sentiment adds value on it's own, it worsens the accuracy when paired with the topic predictions. The topic predictions on the other hand seem to be a rather potent measure of veracity.

TABLE VI: Weka J48 (50 leaves) results of topic and avg. sentiment data

Input data	Correct (%)
only avg. sentiment	54.74%
topic prediction + average sentiment	77.53%
only topic prediction	77.68%

H. Lab 08 – Model combination

The goal of this final lab is to combine the efforts of lab 5 and 7. The intention of the topic modelling in lab 7 was never to combine it with the average article sentiment for data. So a final combined model as described in figure 14 is created. The model creates topic predictions, the LSTM prediction and statistical features for every article and combines them with the veracity label. This data becomes a new row in the final data frame. After about 4 hours of processing the model is done creating the knowledge base. It can be taken to the Weka explorer to create the machine learning model that will compare new data to the existing information.

Sampling different machine learning algorithms in Weka J48 came out on top again. J48 is a decision tree algorithm. It achieved 79.8% correctly classified instances in 10-fold training. The model scores 2.2% higher than topics modelling alone. Again the code can be found in appendix J and on the project GitHub.

VII. EVALUATION

A. Requirements revisited

The evaluation's first assessment is whether the model achieved the early requirements set out in table II.

1. *The model can extract knowledge from raw information.* A liberal view of knowledge was adapted: an article's sentiment sequence and statistics along with its distribution over topics. This knowledge theoretically shows the stance of the author towards the topic. It is not as nuanced as a sentence level stance detection with sentiment or using triples to represent facts, but it did work.

2. *The model has a knowledge base.* With the knowledge definition of requirement 1, this requirement is a clear success. All 44,000+ articles were processed to form a knowledge base.

3. *The model can compare new knowledge to the knowledge base.* This requirement was achieved using Weka. The Weka model for comparison was easily moved from the explorer into python, where it was able to judge new articles.

4. *The model can conclude a comparison.* The model delivers a truth of fake news verdict from the comparison. However, the verdict could be improved by including an uncertainty category. Since the comparison model uses a decision tree, there is no scale between true and fake, it always acts with certainty, even when it should not be.

5. *The model should be more accurate than the baseline.* The model did better than the baseline. From 52% to 79% is a decent improvement.

6. *The model can improve an existing model.* This was never explicitly attempted. The sentiment part of the knowledge did marginally improve only topic detection. So theoretically, the model, parts, or principles can be implemented in a different model. Nevertheless, the requirement remains unproven.

7. *The model can process information in a timely manner.* Although a timely manner is a vague requirement; the model takes a little under half a second to reach a verdict about a new article. Half a second is very usable in single-user applications. However, for processing large amounts of data for a team of researchers, this is inadequate. Although, in lab 8 – model combination, little attention was paid to batch optimisation or vectorisation, so there is probably something to gain.

B. F1-score

A standard measure for performance in classification problems is the F1 score. Instance classification accuracy, used in the document previously is not the best measure for performance. The F1 score instead

takes false positives (FP) and false negatives (FN) into account better. The formula to calculate it is:

$$F_1 = \frac{\text{TruePositives}}{TP + \frac{1}{2}(FP + FN)}$$

When plugging in the false positives and negatives as can be seen in the confusion matrix, table VII an F1 score of 0.798% is discovered. This is not surprising as the false positive and negative categories are of roughly the same size.

TABLE VII: Confusion matrix final model

		Classified	
		Fake	True
Actually	Fake	19595	3886 (FP)
	True	5181 (FN)	16236

C. Case examination

A case examination was conducted where random articles were sampled and the model's verdict was studied. An observation was that the model is usually not very sure about the topics, but is unsure in the right direction. The added sentiment statistics play a small role but are sometimes used as a tie-breaker when the topic and LSTM prediction did were not sufficient for a verdict. A deeper analysis of topics using visualisation methods could have given larger insights but was not achievable within the project's timeframe.

VIII. CONCLUSIONS

This paper proposes a sentiment approach to fake news classification. Following a ground truth theory, an attempt is made to use sentiment data as a way to define an author's stance towards a topic. In the final stages it achieved a 80% accuracy. When the project was started, accuracies were in the neighbourhood of 95% [55]. At the time of writing, papers have been published achieving scores of 99.8% and higher [56, 57]. The results of this project fall somewhat flat in comparison to newer state of the art models. Nevertheless, to the author's best knowledge, this is the first time sentiment has been used as an attempt towards stance detection for fake news. Although [58] suggests that sentiment sequences may be a weak indicator of credibility, this project managed to use them to add a small amount of value.

IX. FUTURE WORK

The ground truth approach, especially a well-developed, nuanced one may deliver much more value than the current state of the art black-box models. Suppose a ground truth model can give a reason why an article or a part of an article is risky. That opens up a path to better handling fake news, even if the ground truth model's accuracy lies lower

than 100%. Explainable machine learning has been attracting more attention, and the fake news problem could benefit from it.

The sentiment approach could be refined down to sentence or claim level as a step in the right direction. Alternatively, triple detection improvements could be a big step towards a ground truth model that can offer human-interpretable decisions.

ACKNOWLEDGEMENTS

The author wants to thank Andreas Kamilaris, Chantal Vriens, Job Zwiers, Wendy Tollenaar and Alexandria Poole for their continued support, input and inspiration.

REFERENCES

- [1] United Nations Development Programme, "Responding to COVID-19 Information Pollution — UNDP," no. May, 2020. [Online]. Available: <https://www.undp.org/content/undp/en/home/librarypage/democratic-governance/responding-to-covid-19-information-pollution.html>
- [2] R. Scherer and F. Siddiq, "The relation between students' socioeconomic status and ICT literacy: Findings from a meta-analysis," *Computers and Education*, 2019.
- [3] S. Vosoughi, D. Roy, and S. Aral, "The spread of true and false news online," *Science*, vol. 359, no. 6380, pp. 1146–1151, 3 2018. [Online]. Available: <http://science.sciencemag.org/>
- [4] "Democracy — United Nations." [Online]. Available: <https://www.un.org/en/sections/issues-depth/democracy/index.html>
- [5] "India buzzes with fake news of 'civil war' in Pakistan - BBC News." [Online]. Available: <https://www.bbc.com/news/world-asia-54649302>
- [6] "List of unproven methods against COVID-19 - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/List_of_unproven_methods_against_COVID-19
- [7] J. L. Egelhofer and S. Lecheler, "Fake news as a two-dimensional phenomenon: a framework and research agenda," 2019.
- [8] A. Gelfert, "Fake news: A definition," *Informal Logic*, 2018.
- [9] M. Verstraete, D. E. Bambauer, and J. R. Bambauer, "Identifying and Countering Fake News," *SSRN Electronic Journal*, 2018.
- [10] D. Southard, "Code and Other Laws of Cyberspace," 2001.
- [11] N. J. Conroy, V. L. Rubin, and Y. Chen, "Automatic deception detection: Methods for finding fake news," *Proceedings of the Association for Information Science and Technology*, 2015.
- [12] J. K. Burgoon, J. P. Blair, T. Qin, and J. F. Nunamaker, "Detecting deception through linguistic analysis," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2003.
- [13] G. Gravanis, A. Vakali, K. Diamantaras, and P. Karadais, "Behind the cues: A benchmarking study for fake news detection," *Expert Systems with Applications*, 2019.
- [14] V. Rubin, N. Conroy, Y. Chen, and S. Cornwell, "Fake News or Truth? Using Satirical Cues to Detect Potentially Misleading News," 2016.
- [15] M. Tambuscio, D. F. M. Oliveira, G. L. Ciampaglia, and G. Ruffo, "Network segregation in a model of misinformation and fact-checking," *Journal of Computational Social Science*, 2018.
- [16] S. Tschiatschek, A. Singla, M. Gomez Rodriguez, A. Merchant, and A. Krause, "Fake News Detection in Social Networks via Crowd Signals," in *The Web Conference 2018 - Companion of the World Wide Web Conference, WWW 2018*, 2018.
- [17] M. L. Della Vedova, E. Tacchini, S. Moret, G. Ballarin, M. Dipierro, and L. De Alfaro, "Automatic online fake news detection combining content and social signals," in *Conference of Open Innovation Association, FRUCT*, 2018.
- [18] N. Ruchansky, S. Seo, and Y. Liu, "CSI: A hybrid deep model for fake news detection," in *International Conference on Information and Knowledge Management, Proceedings*, 2017.
- [19] J. Kim, B. Tabibian, A. Oh, B. Schölkopf, and M. Gomez-Rodriguez, "Leveraging the crowd to detect and reduce the spread of fake news and misinformation," in *WSDM 2018 - Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, 2018.
- [20] J. A. Braun and J. L. Eklund, "Fake News, Real Money: Ad Tech Platforms, Profit-Driven Hoaxes, and the Business of Journalism," *Digital Journalism*, 2019.
- [21] W. H. Dutton and L. Fernandez, "How Susceptible Are Internet Users?" *SSRN Electronic Journal*, 2019.
- [22] J. Kirchner and C. Reuter, "Countering Fake News: A Comparison of Possible Solutions Regarding User Acceptance and Effectiveness," *Proceedings of the ACM: Human Computer Interaction (PACM): Computer-Supported Cooperative Work and Social Computing*, 2020.
- [23] J. M. Burkhardt, *Combating Fake News in the Digital Age*, 2017.
- [24] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," 2017.
- [25] K. Clayton, S. Blair, J. A. Busam, S. Forstner, J. Glance, G. Green, A. Kawata, A. Kovvuri, J. Martin, E. Morgan, M. Sandhu, R. Sang, R. Scholz-Bright, A. T. Welch, A. G. Wolff, A. Zhou, and B. Nyhan, "Real Solutions

- for Fake News? Measuring the Effectiveness of General Warnings and Fact-Check Tags in Reducing Belief in False Stories on Social Media," *Political Behavior*, 2019.
- [26] W. Y. Wang, "“Liar, liar pants on fire”: A new benchmark dataset for fake news detection," in *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, 2017.
 - [27] A. Mader and W. Eggink, "A DESIGN PROCESS FOR CREATIVE TECHNOLOGY," Tech. Rep., 2014. [Online]. Available: <https://research.utwente.nl/en/publications/a-design-process-for-creative-technology>
 - [28] "Welcome to Python.org." [Online]. Available: <https://www.python.org/>
 - [29] "Cython: C-Extensions for Python." [Online]. Available: <https://cython.org/>
 - [30] "Project Jupyter — Home." [Online]. Available: <https://jupyter.org/>
 - [31] "FactCheck.org - A Project of The Annenberg Public Policy Center." [Online]. Available: <https://www.factcheck.org/>
 - [32] "PolitiFact." [Online]. Available: <https://www.politifact.com/>
 - [33] "Snopes.com — The definitive fact-checking site and reference source for urban legends, folklore, myths, rumors, and misinformation." [Online]. Available: <https://www.snopes.com/>
 - [34] "International Fact-Checking Network - Poynter." [Online]. Available: <https://www.poynter.org/ifcn/>
 - [35] B. Rath, W. Gao, and J. Srivastava, "Evaluating vulnerability to fake news in social networks: A community health assessment model," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2019*. Association for Computing Machinery, Inc, 8 2019, pp. 432–435.
 - [36] "WordProof Timestamp - More trust, a safer internet." [Online]. Available: <https://wordproof.com/>
 - [37] X. Zhou and R. Zafarani, "A Survey of Fake News: Fundamental Theories, Detection Methods, and Opportunities," *ACM Computing Surveys*, 2020.
 - [38] H. Che, "A Semantic Web primer," *Journal of the American Society for Information Science and Technology*, 2006.
 - [39] G. Schreiber and Y. Raimond, "Rdf 1.1 Primer," 2014.
 - [40] Y. Liu, T. Zhang, Z. Liang, H. Ji, and D. L. McGuinness, "Seq2rDF: An end-to-end application for deriving triples from natural language text," in *CEUR Workshop Proceedings*, 2018.
 - [41] R. Feldman, "Techniques and applications for sentiment analysis: The main applications and challenges of one of the hottest research areas in computer science," pp. 82–89, 4 2013.
 - [42] O. Appel, F. Chiclana, J. Carter, and H. Fujita, "A hybrid approach to the sentiment analysis problem at the sentence level," *Knowledge-Based Systems*, vol. 108, pp. 110–124, 9 2016.
 - [43] "Is More Data Always Better For Building Analytics Models?" [Online]. Available: <https://analyticsindiamag.com/is-more-data-always-better-for-building-analytics-models/>
 - [44] "Comparing sentiment analysis tools — Data Science for Journalism." [Online]. Available: <https://investigate.ai/investigating-sentiment-analysis/comparing-sentiment-analysis-tools/>
 - [45] C. J. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text," Tech. Rep. 1, 5 2014. [Online]. Available: <http://sentic.net/>
 - [46] "The Stanford Natural Language Processing Group." [Online]. Available: <https://nlp.stanford.edu/software/openie.html>
 - [47] G. Angeli, M. J. Premkumar, and C. D. Manning, "Leveraging linguistic structure for open domain information extraction," in *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, 2015.
 - [48] "Home - DBpedia Organization." [Online]. Available: <https://www.dbpedia.org/>
 - [49] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java." [Online]. Available: <https://www.cs.waikato.ac.nz/ml/weka/>
 - [50] L. R. Medsker, "RECURRENT NEURAL Design and Applications," New York, 2001.
 - [51] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *30th International Conference on Machine Learning, ICML 2013*, 2013.
 - [52] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 1998.
 - [53] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, 2000.
 - [54] A. Onan, S. Korukoğlu, and H. Bulut, "LDA-based Topic Modelling in Text Sentiment Classification: An Empirical Analysis," *International Journal of Computational Linguistics and Applications*, 2016.
 - [55] S. Hakak, M. Alazab, S. Khan, T. R. Gadekallu, P. K. R. Maddikunta, and W. Z. Khan, "An ensemble machine learning approach through effective feature extraction to classify fake news," *Future Generation Computer Systems*, 2021.
 - [56] H. Ahmed, I. Traore, and S. Saad, "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10618 LNCS. Springer Verlag, 10 2017, pp. 127–138. [Online]. Available: https://doi.org/10.1007/978-3-319-69155-8_9
 - [57] M. H. Goldani, R. Safabakhsh, and S. Momtazi, "Convolutional neural network with margin loss for fake news detection," *Information Processing and Management*, vol. 58, no. 1, p. 102418, 1 2021.
 - [58] N. Sitaula, C. K. Mohan, J. Grygiel, X. Zhou, and R. Zafarani, "Credibility-based Fake News Detection," pp. 163–182, 11 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-42699-6_9

APPENDIX

Appendix Lab 01 - Sentiment Analysis

<https://investigate.ai/investigating-sentiment-analysis/comparing-sentiment-analysis-tools/>

a demo of sentiment analyses

technique	word source	word selection	scores
NLTK (VADER)	everywhere	hand-picked	internet people, word-by-word
TextBlob	product reviews	hand-picked, mostly adjectives	internet people, word-by-word
TextBlob + NaiveBayesAnalyzer	movie reviews	all words	automatic based on score

```
In [1]: import nltk

In [2]: nltk.download('vader_lexicon')
nltk.download('movie_reviews')
nltk.download('punkt')

[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package movie_reviews to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package movie_reviews is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!

Out[2]: True

In [3]: from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA

sia = SIA()
print(sia.polarity_scores("The corona vaccine contains dangerous chemicals"))
print(sia.polarity_scores("The corona vaccine contains only safe ingredients and is harmless"))
print(sia.polarity_scores("The corona vaccine contains a compound called Formaldehyde"))

{'neg': 0.383, 'neu': 0.617, 'pos': 0.0, 'compound': -0.4767}
{'neg': 0.0, 'neu': 0.62, 'pos': 0.38, 'compound': 0.5994}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}

In [4]: from textblob import TextBlob
from textblob import Blobber
from textblob.sentiments import NaiveBayesAnalyzer

In [5]: print((TextBlob("The corona vaccine contains dangerous chemicals")).sentiment)
print((TextBlob("The corona vaccine contains only safe ingredients and is harmless")).sentiment)
print((TextBlob("The corona vaccine contains a compound called Formaldehyde")).sentiment)

Sentiment(polarity=-0.6, subjectivity=0.9)
Sentiment(polarity=0.25, subjectivity=0.75)
Sentiment(polarity=0.0, subjectivity=0.0)

In [6]: posneg = Blobber(analyzer=NaiveBayesAnalyzer())

print((posneg("The corona vaccine contains dangerous chemicals")).sentiment)
print((posneg("The corona vaccine contains only safe ingredients and is harmless")).sentiment)
print((posneg("The corona vaccine contains a compound called Formaldehyde")).sentiment)

Sentiment(classification='neg', p_pos=0.26831221732575683, p_neg=0.7316877826742444)
Sentiment(classification='pos', p_pos=0.6241247363042428, p_neg=0.3758752636957584)
Sentiment(classification='neg', p_pos=0.40799306089467274, p_neg=0.5920069391053251)

In [7]: import pandas as pd
pd.set_option("display.max_colwidth", 200)

df = pd.DataFrame({'content': [
    "I love love love love this kitten",
    "I hate hate hate hate this keyboard",
    "I'm not sure how I feel about toast",
    "Did you see the baseball game yesterday?",
    "The package was delivered late and the contents were broken",
    "Trashy television shows are some of my favorites",
    "I'm seeing a Kubrick film tomorrow, I hear not so great things about it.",
    "I find chirping birds irritating, but I know I'm not the only one",
    "-----",
    "The corona vaccine contains dangerous chemicals",
    "The corona vaccine contains only safe ingredients and is harmless",
    "The corona vaccine contains a compound called Formaldehyde",
]})
df
```

	content
0	I love love love love this kitten
1	I hate hate hate hate this keyboard
2	I'm not sure how I feel about toast

content	
3	Did you see the baseball game yesterday?
4	The package was delivered late and the contents were broken
5	Trashy television shows are some of my favorites
6	I'm seeing a Kubrick film tomorrow, I hear not so great things about it.
7	I find chirping birds irritating, but I know I'm not the only one
8	-----
9	The corona vaccine contains dangerous chemicals
10	The corona vaccine contains only safe ingredients and is harmless

```
In [8]: def get_scores(content):
        blob = TextBlob(content)
        nb_blob = posneg(content)
        sia_scores = sia.polarity_scores(content)

        return pd.Series({
            'content': content,
            'textblob': blob.sentiment.polarity,
            'textblob_bayes': nb_blob.sentiment.p_pos - nb_blob.sentiment.p_neg,
            'nltk': sia_scores['compound'],
        })

scores = df.content.apply(get_scores)
scores.style.background_gradient(cmap='RdYlGn', axis=None, low=0.4, high=0.4)
```

	content	textblob	textblob_bayes	nltk
0	I love love love love this kitten	0.500000	-0.087933	0.957100
1	I hate hate hate hate this keyboard	-0.800000	-0.214151	-0.941300
2	I'm not sure how I feel about toast	-0.250000	0.394659	-0.241100
3	Did you see the baseball game yesterday?	-0.400000	0.613050	0.000000
4	The package was delivered late and the contents were broken	-0.350000	-0.574270	-0.476700
5	Trashy television shows are some of my favorites	0.000000	0.040076	0.421500
6	I'm seeing a Kubrick film tomorrow, I hear not so great things about it.	0.800000	0.717875	-0.629600
7	I find chirping birds irritating, but I know I'm not the only one	-0.200000	0.257148	-0.250000
8	-----	0.000000	0.000000	0.000000
9	The corona vaccine contains dangerous chemicals	-0.600000	-0.463376	-0.476700
10	The corona vaccine contains only safe ingredients and is harmless	0.250000	0.248249	0.599400
11	The corona vaccine contains a compound called Formaldehyde	0.000000	-0.184014	0.000000

Appendix Lab 02 - Triple Extraction

using stanford's OpenIE

```
In [17]: from openie import StanfordOpenIE # first get torch from pytorch.org/get-started/locally

import os

import networkx as nx
import matplotlib.pyplot as plt
from IPython.display import Image
from IPython.display import SVG
print('imports done')

imports done
```

```
In [2]: new_keys = {'subject':'S', 'relation':'P', 'object':'O'}
def change_triple_keys(triple):
    return dict((new_keys[key], value) for (key, value) in triple.items())
```

```
In [39]: def generate_text_graph(triples: dict = None, svg_filename: str = 'out/graph.svg'):
graph = nx.MultiDiGraph()
triple_count = len(triples)
for idx, sor in enumerate(triples):
    if idx%round(triple_count/10) == 0:
        print(idx, " ", sor)
        graph.add_edge(sor['subject'], sor['object'], label=sor['relation'])

print(graph)

pos=nx.spring_layout(graph)

p=nx.drawing.nx_pydot.to_pydot(graph)
p.write_svg(svg_filename)
return svg_filename
```

```
In [4]: sample_sentences = ["The corona vaccine contains dangerous chemicals",
                             "The corona vaccine contains only safe ingredients and is harmless",
                             "The corona vaccine contains a compound called Formaldehyde"]

with StanfordOpenIE() as client:
    for sentence in sample_sentences:
        print('\n__ %s.' % sentence)
        for triple in client.annotate(sentence):
            print('|-', change_triple_keys(triple))
```

__ The corona vaccine contains dangerous chemicals.

Starting server with command: java -Xmx8G -cp C:\Users\emiel\stanfordnlp_resources\stanford-corenlp-full-2018-10-05/* edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000 -timeout 60000 -threads 5 -maxCharLength 100000 -quiet True -serverProperties corenlp_server-518c64b3d6e0404e.props -preload openie

```
| - {'S': 'corona vaccine', 'P': 'contains', 'O': 'chemicals'}
| - {'S': 'corona vaccine', 'P': 'contains', 'O': 'dangerous chemicals'}
```

__ The corona vaccine contains only safe ingredients and is harmless.

```
| - {'S': 'corona vaccine', 'P': 'contains', 'O': 'safe ingredients'}
| - {'S': 'corona vaccine', 'P': 'contains', 'O': 'ingredients'}
| - {'S': 'corona vaccine', 'P': 'contains', 'O': 'only ingredients'}
| - {'S': 'corona vaccine', 'P': 'contains', 'O': 'only safe ingredients'}
```

__ The corona vaccine contains a compound called Formaldehyde.

```
| - {'S': 'compound', 'P': 'called', 'O': 'Formaldehyde'}
```

```
In [5]: txt_path = os.path.join('sources', 'article_corona_vaccine.txt')
extracted_triples = list()
with StanfordOpenIE() as client:
    with open(txt_path, 'r', encoding='utf-8') as r:
        text = r.read().replace('\n', ' ').replace('\r', '')
        extracted_triples = client.annotate(text, simple_format=True)
        print('\n',text[:500:],'\n', ) # to check the formatting
        print('Found %s triples in the corpus.' % len(extracted_triples))
```

Starting server with command: java -Xmx8G -cp C:\Users\emiel\stanfordnlp_resources\stanford-corenlp-full-2018-10-05/* edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000 -timeout 60000 -threads 5 -maxCharLength 100000 -quiet True -serverProperties corenlp_server-3c94889c679d4596.props -preload openie

Developing a vaccine is a long-term process. It is difficult to say when there will be an effective vaccine against the novel coronavirus. It takes a long time to develop a vaccine for a new infectious disease, usually as long as 5 to 10 years. Every effort is being made to accelerate development of a vaccine for COVID-19. More than 100 vaccines are in development. A total of more than 100 different vaccines for SARS-CoV-2 are under development. A small number of them have reached the stage of

Found 429 triples in the corpus.

```
In [41]: SVG(generate_text_graph(extracted_triples))
print('done')
```

```
0  {'subject': 'It', 'relation': 'be', 'object': 'when vaccine against novel coronavirus'}
43  {'subject': 'phases', 'relation': 'have', 'object': 'Only when have successfully completed'}
86  {'subject': 'we', 'relation': 'are protected After', 'object': 'this'}
129  {'subject': 'S protein', 'relation': 'is attached to', 'object': 'exterior of virus'}
172  {'subject': 'RNA vaccines', 'relation': 'add', 'object': 'piece to immune cells in our body'}
215  {'subject': 'RNA vaccines', 'relation': 'add', 'object': 'piece of material to specific immune cells in our body'}
```

```
258    {'subject': 'immune cells', 'relation': 'are', 'object': 'why also referred to as cells'}
301    {'subject': 'lymphocytes', 'relation': 'neutralise', 'object': 'bacteria'}
344    {'subject': 'category', 'relation': 'are', 'object': 'adenoviruses'}
387    {'subject': 'Royal Netherlands Society', 'relation': 'keep track on', 'object': 'basis of progress on development i
n trials'}
```

Appendix Lab 03 - Sentiment Model

working out the sentiment sequence and statistical data

```
In [1]: import os
from ArticleModule import Article

import pandas as pd
pd.set_option("display.precision", 3)

import matplotlib.pyplot as plt
%matplotlib inline
plt.close('all')
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
```

```
In [2]: test_article = None

txt_path = os.path.join('sources', 'article_corona_vaccine.txt')
with open(txt_path, 'r', encoding='utf-8') as r:
    text = r.read().replace('\n', ' ').replace('\r', '')
    test_article = Article(text, "Corona Test Article", 1)
```

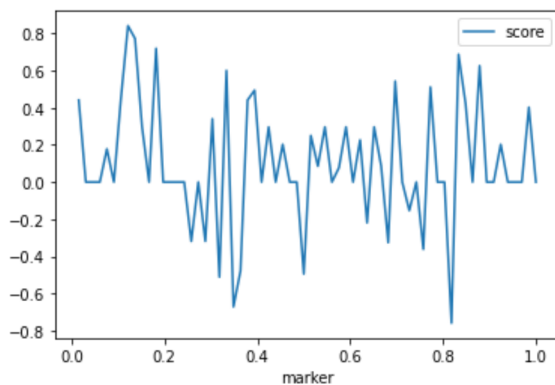
```
In [ ]: print(test_article.article_text)
```

```
In [4]: print(test_article.article_title)
print(test_article.stats)
```

```
Corona Test Article
count    66.000
mean     0.098
std      0.333
min     -0.758
25%      0.000
50%      0.000
75%      0.296
max       0.840
Name: score, dtype: float64
```

```
In [5]: fingerprint = test_article.data.plot(y='score', x='marker')
fingerprint
```

```
Out[5]: <AxesSubplot:xlabel='marker'>
```



```
In [6]: dataset_path = os.path.join("sources", "snopes_checked_v02.csv")
dataset_load = pd.read_csv(dataset_path, encoding='ANSI')
dataset_load.head()
```

	fact_rating_phase1	snopes_url_phase1	article_title_phase1	article_category_phase1	article_date_phase1	article_claim_phase1	article_origin_url_phase1
0	mixture	https://www.snopes.com/fact-check/elizabeth-wa...	Did Elizabeth Warren Lie About Her Native Amer...	Politics Politicians	30-Oct-14	Elizabeth Warren lives in a multi-million-doll...	https://www.bostonglobe.com/metro/2012/01/13/e...
1	mixture	https://www.snopes.com/fact-check/the-company-...	Animal Testing Resumes for Avon, Mary Kay, and...	Critter Country Media Matters	31-Mar-15	Cosmetic companies Avon, Mary Kay and Este Lau...	http://www.thedogfiles.com/2012/02/27/animal-t...
2	mixture	https://www.snopes.com/fact-check/hillary-clin...	Did Hillary Clinton Tell FBIs Mueller to Deliv...	Politics Conspiracy Theories	21-Jun-17	Then-Secretary of State Clinton ordered then-F...	https://www.intellihub.com/hillary-clinton-req...
3	mixture	https://www.snopes.com/fact-check/light-fingered/	Dangers of CFL Mercury Light Bulbs	Medical	02-May-07	Energy-saving light bulbs release dangerous a...	https://web.archive.org/web/20071003140931/htt...

```

In [7]: dataset = pd.DataFrame({
    'title': dataset_load['article_title_phase2'],
    'body': dataset_load['original_article_text_phase2'],
    'category': dataset_load['article_category_phase1'],
    'published': dataset_load['publish_date_phase2'],
    'descr_rating': dataset_load['fact_rating_phase1']
})

def apply_rating(content):
    if content in ['mixture', 'mostly false', 'FALSE']:
        return 0
    if content in ['mostly true', 'TRUE']:
        return 1

dataset['rating'] = dataset['descr_rating'].apply(apply_rating)
dataset.head()

```

Out[7]:

		title	body	category	published	descr_rating	rating
0	Elizabeth Warrens salary topped \$700,000 over ...	Elizabeth Warren, the Harvard Law School profe...	Politics Politicians	1/13/12	mixture	0	
1	Animal Testing Resumes For Avon, Mary Kay, And...	As of right now I dont know what kinds of anim...	Critter Country Media Matters	2/27/12	mixture	0	
2	Hillary Clinton requested FBI Dir. Mueller del...	\r\r\r\r\r\r\r Former Secretary of State Hillar...	Politics Conspiracy Theories	6/19/17	mixture	0	
3	Mercury In King County: Crushing Lamps	Crushing Lamps\r\rCrushing and breaking fluore...	Medical	NaN	mixture	0	
4	Uncle Sam's Misguided Children	This was their response: "Logistically, we cou...	Politics Politics	6/15/18	mixture	0	

```

In [8]: articles = []
for index, entry in dataset.iterrows():
    articles.append(Article(entry['body'], entry['title'], entry['rating']))

```

```

In [9]: articles_fake = []
articles_true = []
for article in articles:
    if article.rating:
        articles_true.append(article)
    else:
        articles_fake.append(article)

```

```

In [10]: import random

```

```

In [23]: t = random.sample(articles_true, 2)
f = random.sample(articles_fake, 2)

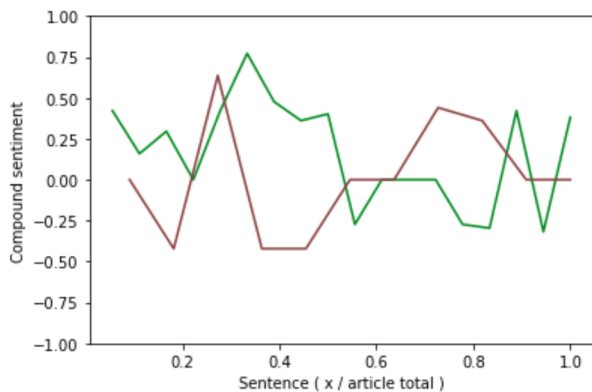
plt.plot(t[0].data.marker, t[0].data.score, '-', color="#90ee90") #light green
plt.plot(t[1].data.marker, t[1].data.score, '-', color="#018b20") #dark green
plt.plot(f[0].data.marker, f[0].data.score, '-', color="#ffcccb") #light red
plt.plot(f[1].data.marker, f[1].data.score, '-', color="#8b4040") #dark red
plt.xlabel("Sentence ( x / article total )")
plt.ylabel("Compound sentiment")
plt.ylim([-1, 1])

```

```

Out[23]: (-1.0, 1.0)

```



```

In [ ]: display(f[0].article_text)

```

```

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
import os
import re
import pandas as pd

#Vars for manual sentence splitter
alphabets = "([A-Za-z])"
prefixes = "(Mr|St|Mrs|Ms|Dr|Prof|Capt|Cpt|Lt|Mt)[.]"
suffixes = "(Inc|Ltd|Jr|Sr|Co)"
starters = "(Mr|Mrs|Ms|Dr|He\s|She\s|It\s|They\s|Their\s|Our\s|We\s|But\s|However\s|That\s|This\s|Wherever)"
acronyms = "([A-Z][.])[A-Z].{1,4}([A-Z])?"
websites = "([.](com|net|org|io|gov|me|edu|nl|be|de))"

```

```

#file for punkt splitter
nltk.download('punkt')

```

```

#file for vader sentiment
nltk.download('vader_lexicon')

```

```

#sia
sia = SIA()

```

```

class Article:

```

```

    def __init__(self, article_text_in, article_title_in, rating_in):

```

```

        self.article_text = article_text_in
        self.article_title = article_title_in
        self.rating = rating_in
        self.prediction = None

```

```

        self.data = pd.DataFrame()

```

```

        self.set_sentences_manual()
        self.set_sentiments()

```

```

        self.stats = self.data['score'].describe()

```

```

        self.set_markers()

```

```

    def set_sentences_manual(self):

```

```

        # splitting sentences more or less manually
        # https://stackoverflow.com/questions/4576077/how-can-i-split-a-text-into-sentences
        # -*- coding: utf-8 -*-
        text = self.article_text
        text = " " + text + " "
        text = text.replace("\n", " ")
        text = re.sub(prefixes, "\\1<prd>", text)
        text = re.sub(websites, "<prd>\\1", text)
        if "Ph.D" in text: text = text.replace("Ph.D.", "Ph<prd>D<prd>")
        text = re.sub("\s" + alphabets + "[.]", "\\1<prd>", text)
        text = re.sub(acronyms + " " + starters, "\\1<stop> \\2", text)
        text = re.sub(alphabets + "[.]" + alphabets + "[.]" + alphabets + "[.]", "\\1<prd>\\2<prd>\\3<prd>", text)
        text = re.sub(alphabets + "[.]" + alphabets + "[.]", "\\1<prd>\\2<prd>", text)
        text = re.sub(" " + suffixes + "[.] " + starters, "\\1<stop> \\2", text)
        text = re.sub(" " + suffixes + "[.]", "\\1<prd>", text)
        text = re.sub(" " + alphabets + "[.]", "\\1<prd>", text)
        if "" in text: text = text.replace(".", "")
        if "\" in text: text = text.replace(".", "\"")
        if "!" in text: text = text.replace("\", "\"!")
        if "?" in text: text = text.replace("?", "\"?")
        if "..." in text: text = text.replace("...", "<prd><prd><prd>")
        text = text.replace(".", "<stop>")
        text = text.replace("?", "<stop>")
        text = text.replace("!", "<stop>")
        text = text.replace("<prd>", ".")

```

Appendix - Article Module

```
sentences = text.split("<stop>")
sentences = sentences[:-1]
sentences = [s.strip() for s in sentences]

self.data = pd.DataFrame({'sentence' : sentences})

return None

def set_sentences_punkt(self):
    self.data = pd.DataFrame({'sentence' : nltk.tokenize.sent_tokenize(self.article_text)})
    return None

def get_scores(self, content, method='VADER'):
    if method == 'VADER':
        scores = sia.polarity_scores(content)['compound'] #list of compound score per sentence
    else:
        scores = None

    return pd.Series({
        'score': scores
    })

def set_sentiments(self):
    self.data['score'] = self.data.sentence.apply(self.get_scores)
    return None

def set_markers(self):
    self.data['marker'] = (self.data.index+1)/self.stats['count']
    return None
```

Appendix Lab 04 - Pre-Processing

In this notebook we will use what we learned in lab 03 about forming the data, the end result is a dataset that we can feed weka to test models on.

So we will:

1. Load the dataset
2. Split the text into sentences
3. Perform sentiment analysis on the sentences
4. Include statistical features
5. Resample the data to 150 points
6. Store it in a row of a dataframe
7. Export it to a csv

```
In [2]: # imports
import os
import pandas as pd
import numpy as np

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sia = SIA()

import nltk
#file for punkt splitter
nltk.download('punkt');
#file for vader sentiment
nltk.download('vader_lexicon');

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"]=20,20
%matplotlib inline
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
```

1: Load the dataset

```
In [3]: dataset_path_true = os.path.join("sources", "ISOT", "True.csv")
dataset_path_fake = os.path.join("sources", "ISOT", "Fake.csv")

dataset_load_true = pd.read_csv(dataset_path_true, encoding='utf-8') # make sure to use the right encoding
dataset_load_fake = pd.read_csv(dataset_path_fake, encoding='utf-8')

display(dataset_load_true.head())
display(dataset_load_fake.head())
```

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017

	title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017

2: Split the text into sentences

in lab 03 we used a big manual function, but for now we will use the nltk tokenizer in this one :)

```
In [4]: def split_sentences(article_text):
        """Takes a string, returns a list of its individual sentences ()"""
        return pd.Series(nltk.tokenize.sent_tokenize(article_text))

sample_sentences = split_sentences(dataset_load_true.text[0])
display(sample_sentences.iloc[[0,1,-2,-1]])

0    WASHINGTON (Reuters) - The head of a conservat...
1    In keeping with a sharp pivot under way among ...
28   The package far exceeded the $44 billion reque...
29           The Senate has not yet voted on the aid.
dtype: object
```

3: Performing sentiment analysis

```
In [5]: def get_scores(text: list, method='VADER'):
        if method == 'VADER':
            scores = text.apply(lambda s: sia.polarity_scores(s)['compound']) #list of compound score per sentence
        else:
            scores = None

        return scores

sample_scores = get_scores(sample_sentences)
print(sample_scores.iloc[[0,1,-2,-1]])

0    -0.4215
1    -0.4588
28    0.0000
29    0.0000
dtype: float64
```

4: Include statistical features

```
In [6]: def get_stats(scores):
        stats = scores.describe()
        return stats

sample_stats = get_stats(sample_scores)
print(sample_stats)

count      30.000000
mean        0.076147
std         0.402857
min        -0.636900
25%        -0.077025
50%         0.000000
75%         0.318200
max         0.906200
dtype: float64
```

5: Resample the data

```
In [7]: def resample_to_x(scores, scores_len=0, x = 100, to_dict = False):
        scores_len = len(scores)
        if scores_len <= x: #resample to higher (or same) res
            if scores_len > 1:
                slope = (x - 1) / (scores_len - 1)
                new_indices = ['p_'+f'{i:03}' for i in range(int(scores_len))]

                new_scores = scores[:]
                new_scores.index = new_indices

                new_series = pd.Series(index = ['p_'+f'{i:03}' for i in range(int(x))], dtype='float64')

                output = new_scores.combine_first(new_series)
                output = output.interpolate()
            elif scores_len == 1:
                output = pd.Series(data = scores[0], index = ['p_'+f'{i:03}' for i in range(int(x))], dtype='float64')
            else:
                output = pd.Series(data = None, index = ['p_'+f'{i:03}' for i in range(int(x))], dtype='float64')

        elif scores_len > x: #downscale ;(
            y = int(np.ceil(len(scores)/x)) # bin size

            new_scores = [np.mean(i) for i in np.array_split(np.array(scores), x)]
            output = pd.Series(data = new_scores, index = ['p_'+f'{i:03}' for i in range(int(x))], dtype='float64')

        if not to_dict:
            return output
        else:
            return output.to_dict()

sample_interp = resample_to_x(sample_scores, sample_stats['count'])
```

```

In [16]: fig, axes = plt.subplots(nrows=3, ncols=1)
plt.subplots_adjust(hspace=0.8)

sample_scores.plot(ax=axes[0]);
axes[0].set_title('Original Scores');
axes[0].set_ylim([-1, 1])

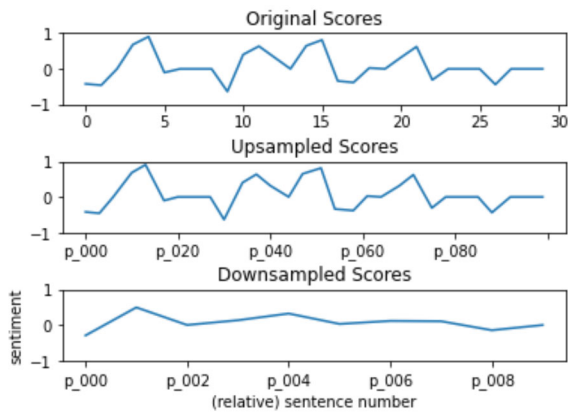
sample_interp.plot(ax=axes[1]);
axes[1].set_title('Upsampled Scores');
axes[1].set_ylim([-1, 1])

downsample_interp = resample_to_x(sample_scores, sample_stats['count'], x=10)
downsample_interp.plot(ax=axes[2]);

axes[2].set_title('Downsampled Scores');
plt.xlabel("(relative) sentence number")
plt.ylabel("sentiment")
plt.ylim([-1, 1])

```

Out[16]: (-1.0, 1.0)



```

In [18]: s1 = resample_to_x(get_scores(split_sentences(dataset_load_true.text[0])))
s2 = resample_to_x(get_scores(split_sentences(dataset_load_true.text[3])))
s3 = resample_to_x(get_scores(split_sentences(dataset_load_true.text[10])))

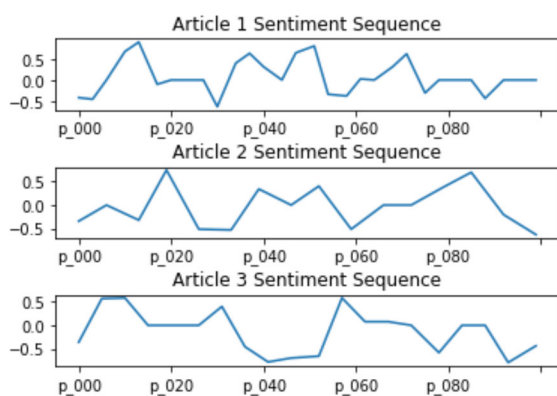
fig, axes = plt.subplots(nrows=3, ncols=1)
plt.subplots_adjust(hspace=0.8)

s1.plot(ax=axes[0]);
axes[0].set_title('Article 1 Sentiment Sequence');

s2.plot(ax=axes[1]);
axes[1].set_title('Article 2 Sentiment Sequence');

s3.plot(ax=axes[2]);
axes[2].set_title('Article 3 Sentiment Sequence');

```



6: Create the dataframe

```
In [8]: def get_all(text, veracity: str, resample_size: int = 200):
    try:
        # should all be dicts with the appropriate name
        sentences = split_sentences(text) # list
        scores = get_scores(sentences) # list but ok, will be
        stats = get_stats(scores) # dict yes!
        scores_interp = resample_to_x(scores, stats['count'], resample_size, to_dict=True) # dict si (p_000 through p_res
        v = {'veracity': veracity}
        v.update(stats)
        v.update(scores_interp)
        return pd.Series(v)
    except Exception as e:
        print("ERROR on text: \n"+text+"\n\n",stats['count'])
        raise e

# print(get_all(dataset_load_true.text[0], 1))
```

```
In [9]: %%timeit

# this cell is for verifying that the method at least works on a small scale (8 instances)
# much quicker to debug than large scale (40k+ instances) ;(

resample_size = 200

cols = ['veracity', 'count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max']
cols.extend(['p_'+f'{n:03}' for n in range(resample_size)])

miniset_true = dataset_load_true.head()
miniset_fake = dataset_load_fake.head()

miniset_prepro = pd.DataFrame(columns = cols)
miniset_prepro[cols] = miniset_true.text.apply(get_all, args=('T', resample_size));
miniset_prepro = pd.concat([miniset_prepro, (miniset_fake.text.apply(get_all, args=('F', resample_size)))], ignore_index=

miniset_prepro.iloc[[0,1,-2,-1]]
miniset_prepro.to_csv(os.path.join('out', 'mini_truefake.csv'), index=False)

185 ms ± 3.87 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
In [ ]: %%time # takes about 6 minutes on my device so I recommend skipping this cell :)

resample_size = 200

cols = ['veracity', 'count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max']
cols.extend(['p_'+f'{n:03}' for n in range(resample_size)])

dataset_prepro = pd.DataFrame(columns = cols)

print("We'll give it a shot!")
dataset_prepro[cols] = dataset_load_true.text.apply(get_all, args=('T', resample_size));
print("Woah, we're halfway there!")
dataset_prepro = pd.concat([dataset_prepro, (dataset_load_fake.text.apply(get_all, args=('F', resample_size)))], ignore_i
print("Woah, livin' on a prayer")

dataset_prepro = dataset_prepro[dataset_prepro['count'] != 0.0]

display(dataset_prepro)
```

```
In [21]: display(dataset_prepro.head())
```

	veracity	count	mean	std	min	25%	50%	75%	max	p_000	...	p_190	p_191	p_192	p_193	p_194	p_195
0	T	30.0	0.076147	0.402857	-0.6369	-0.077025	0.0000	0.31820	0.9062	-0.4215	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	T	21.0	0.018648	0.486143	-0.8625	-0.292400	0.0772	0.34000	0.7531	-0.6908	...	-0.240530	-0.188660	-0.136790	-0.084920	-0.033050	0.018800
2	T	17.0	-0.010176	0.450475	-0.6808	-0.273200	0.0000	0.38180	0.7269	0.6124	...	-0.209477	-0.261846	-0.314215	-0.366585	-0.418954	-0.471300
3	T	16.0	-0.029581	0.438841	-0.6249	-0.382650	0.0000	0.35045	0.7430	-0.3400	...	-0.353229	-0.383414	-0.413600	-0.443786	-0.473971	-0.504100
4	T	40.0	0.047153	0.327357	-0.6124	-0.156025	0.0000	0.26705	0.6249	0.0000	...	-0.163920	-0.109280	-0.054640	0.000000	0.000000	0.000000

5 rows × 209 columns

7: Export to CSV

```
In [ ]: dataset_prepro.to_csv(os.path.join('out', 'truefake.csv'), index=False)
```

Appendix Lab 05 - Recurrent Neural Network

More specifically a Long Short Term Memory one

```
In [1]: import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, Masking, Embedding
```

```
In [2]: # imports
import os
import pandas as pd
import numpy as np

import random

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sia = SIA()

import nltk
#file for punkt splitter
nltk.download('punkt');
#file for vader sentiment
nltk.download('vader_lexicon');

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"]=20,20
%matplotlib inline
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
```

Data loading/collection

```
In [3]: series_len = 100
mask_value = -10
max_len = 0

def text_to_sentiments(text):
    sentences = nltk.tokenize.sent_tokenize(text)
    scores = [sia.polarity_scores(s)['compound'] for s in sentences] #list of compound score per sentence

    if len(scores) < series_len:
        t = series_len - len(scores)
        scores = np.pad(scores, (t, 0), mode='constant', constant_values=mask_value)
        scores = scores.reshape((series_len, 1))
    else:
        scores = np.array(scores[(-1*series_len):])
        scores = scores.reshape((series_len, 1))

    return scores
```

```
In [4]: %%time

df = None
try:
    with open(os.path.join('out', 'truefake_series.pickle'), 'rb') as f: df = pickle.load(f)
    print('Succesfully loaded data')

except FileNotFoundError:
    print('ERROR: Couldn\'t find data, will start building it now...')
    dataset_path_true = os.path.join("sources", "ISOT", "True.csv")
    dataset_path_fake = os.path.join("sources", "ISOT", "Fake.csv")

    dataset_load_true = pd.read_csv(dataset_path_true, encoding='utf-8') # make sure to use the right encoding
    dataset_load_fake = pd.read_csv(dataset_path_fake, encoding='utf-8')

    df_T = pd.DataFrame(columns = ['scores', 'veracity'])
    df_F = pd.DataFrame(columns = ['scores', 'veracity'])

    df_T['scores'] = dataset_load_true.text.apply(text_to_sentiments)
    df_T['veracity'] = 1

    df_F['scores'] = dataset_load_fake.text.apply(text_to_sentiments)
    df_F['veracity'] = 0

    df = pd.concat([df_T, df_F], ignore_index=True)
    df = df[df['scores'].map(lambda d: len(d) > 0)]
    df.reset_index(drop = True, inplace=True)

    df = pd.concat([df_T, df_F], ignore_index=True)
    df = df[df['scores'].map(lambda d: len(d) > 0)]
    df.reset_index(drop = True, inplace=True)

    df.to_pickle(os.path.join('out', 'truefake_series.pickle'))

finally:
    display(df)
```

ERROR: Couldn't find data, will start building it now...

	scores	veracity
0	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	1
1	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	1
2	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	1
3	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	1
4	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	1
...	...	...
44893	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	0
44894	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	0
44895	[[-0.0772], [-0.9646], [-0.9522], [0.4272], [0...	0
44896	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	0
44897	[[-10.0], [-10.0], [-10.0], [-10.0], [-10.0], ...	0

44898 rows × 2 columns

Wall time: 4min 43s

Splitting data

```
In [5]: def train_test_split(df_in, test_size, shuffle=True):

    if shuffle:
        df_in = df_in.sample(frac=1).reset_index(drop=True)

    mask_test = np.random.rand(len(df_in)) < test_size # the sample is large enough to probably not care about proper div

    train = df_in[~mask_test]
    test = df_in[mask_test]

    X_train = np.array(train['scores'].tolist())
    y_train = np.array(train['veracity'].tolist())

    X_test = np.array(test['scores'].tolist())
    y_test = np.array(test['veracity'].tolist())

    return (X_train, y_train), (X_test, y_test)
```

```
In [6]: # z = np.array([0.1,0.2,0.3])
# z = z.reshape(3, 1)
# print(z,'\n')
# x = pd.DataFrame({'A': [z,z,z,z]})
# print(x,'\n')
# y = np.array(x['A'].to_list())
# print(y,'\n',y.shape)
```

```
In [7]: np.random.seed(100)
random.seed(100)

# # turn T/F labels into 1/0
# d = {'T': 1, 'F':0}
# df['veracity'] = df['veracity'].map(d)

(X_train, y_train), (X_test, y_test) = train_test_split(df, test_size = 0.4)

print("train: ",len(X_train), len(y_train),
      "\ntest: ",len(X_test), len(y_test),
      "\ntest/total: {:.3f} %".format(len(X_test)/(len(X_test)+len(X_train)))
    )

train:  26825 26825
test:   18073 18073
test/total: 0.403 %
```

Building the model: 100-100-100 w/ dropout

```
In [8]: from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, Masking, Embedding
```

```
In [16]: model = None

model = Sequential()

model.add(Masking(mask_value=mask_value,
                  input_shape=(series_len, 1)))

model.add(LSTM(100))

model.add(LSTM(100,return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(100))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
masking_1 (Masking)	(None, 100, 1)	0
lstm_2 (LSTM)	(None, 100, 100)	40800
dropout (Dropout)	(None, 100, 100)	0
lstm_3 (LSTM)	(None, 100, 100)	80400
dropout_1 (Dropout)	(None, 100, 100)	0
lstm_4 (LSTM)	(None, 100)	80400
dropout_2 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 1)	101
Total params: 201,701		
Trainable params: 201,701		
Non-trainable params: 0		
None		

```
In [17]: history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=64)
```

```
Epoch 1/20
420/420 [=====] - 149s 354ms/step - loss: 0.6874 - accuracy: 0.5457 - val_loss: 0.6834 - val_acc
uracy: 0.5798
Epoch 2/20
420/420 [=====] - 146s 347ms/step - loss: 0.6787 - accuracy: 0.5778 - val_loss: 0.6785 - val_acc
uracy: 0.5745
Epoch 3/20
```

```

420/420 [=====] - 151s 360ms/step - loss: 0.6713 - accuracy: 0.5860 - val_loss: 0.6669 - val_acc
uracy: 0.5961
Epoch 4/20
420/420 [=====] - 147s 350ms/step - loss: 0.6648 - accuracy: 0.5979 - val_loss: 0.6607 - val_acc
uracy: 0.6033
Epoch 5/20
420/420 [=====] - 147s 351ms/step - loss: 0.6581 - accuracy: 0.6052 - val_loss: 0.6568 - val_acc
uracy: 0.6064
Epoch 6/20
420/420 [=====] - 149s 355ms/step - loss: 0.6556 - accuracy: 0.6092 - val_loss: 0.6575 - val_acc
uracy: 0.6080
Epoch 7/20
420/420 [=====] - 149s 355ms/step - loss: 0.6520 - accuracy: 0.6158 - val_loss: 0.6566 - val_acc
uracy: 0.6069
Epoch 8/20
420/420 [=====] - 148s 352ms/step - loss: 0.6496 - accuracy: 0.6173 - val_loss: 0.6524 - val_acc
uracy: 0.6190
Epoch 9/20
420/420 [=====] - 150s 357ms/step - loss: 0.6472 - accuracy: 0.6253 - val_loss: 0.6505 - val_acc
uracy: 0.6221
Epoch 10/20
420/420 [=====] - 150s 357ms/step - loss: 0.6451 - accuracy: 0.6275 - val_loss: 0.6486 - val_acc
uracy: 0.6261
Epoch 11/20
420/420 [=====] - 149s 354ms/step - loss: 0.6425 - accuracy: 0.6326 - val_loss: 0.6454 - val_acc
uracy: 0.6302
Epoch 12/20
420/420 [=====] - 148s 352ms/step - loss: 0.6394 - accuracy: 0.6326 - val_loss: 0.6515 - val_acc
uracy: 0.6203
Epoch 13/20
420/420 [=====] - 150s 358ms/step - loss: 0.6374 - accuracy: 0.6371 - val_loss: 0.6443 - val_acc
uracy: 0.6300
Epoch 14/20
420/420 [=====] - 150s 358ms/step - loss: 0.6353 - accuracy: 0.6392 - val_loss: 0.6425 - val_acc
uracy: 0.6281
Epoch 15/20
420/420 [=====] - 149s 354ms/step - loss: 0.6330 - accuracy: 0.6410 - val_loss: 0.6416 - val_acc
uracy: 0.6286
Epoch 16/20
420/420 [=====] - 150s 356ms/step - loss: 0.6308 - accuracy: 0.6398 - val_loss: 0.6412 - val_acc
uracy: 0.6307
Epoch 17/20
420/420 [=====] - 150s 357ms/step - loss: 0.6330 - accuracy: 0.6394 - val_loss: 0.6395 - val_acc
uracy: 0.6377
Epoch 18/20
420/420 [=====] - 148s 351ms/step - loss: 0.6276 - accuracy: 0.6457 - val_loss: 0.6381 - val_acc
uracy: 0.6366
Epoch 19/20
420/420 [=====] - 149s 355ms/step - loss: 0.6258 - accuracy: 0.6483 - val_loss: 0.6318 - val_acc
uracy: 0.6438
Epoch 20/20
420/420 [=====] - 151s 359ms/step - loss: 0.6232 - accuracy: 0.6520 - val_loss: 0.6325 - val_acc
uracy: 0.6444

```

```

In [11]: scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

```

Accuracy: 63.84%

```

In [15]: # Thanks Jason Brownlee
# https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

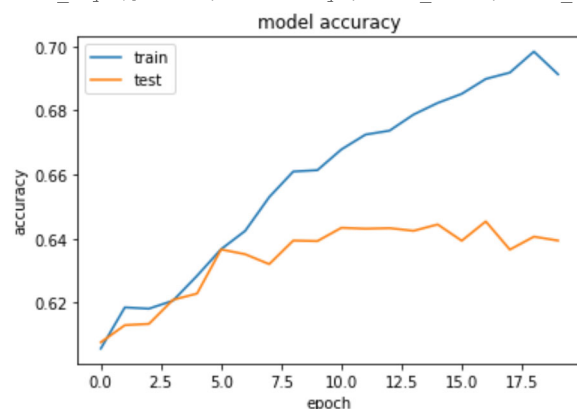
```

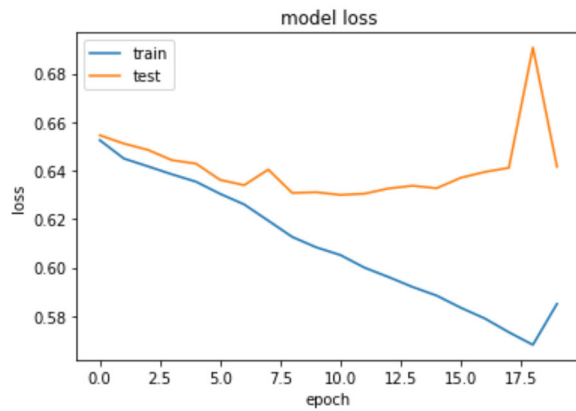
```

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

```





Building the model: 100-1

```
In [27]: model = None

model = Sequential()

model.add(Masking(mask_value=mask_value,
                  input_shape=(series_len, 1)))

model.add(LSTM(100))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
masking_3 (Masking)	(None, 100, 1)	0
lstm_6 (LSTM)	(None, 100, 100)	40800
dense_3 (Dense)	(None, 100, 1)	101
Total params: 40,901		
Trainable params: 40,901		
Non-trainable params: 0		
None		

```
In [24]: history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=30, batch_size=64)
```

```
Epoch 1/30
420/420 [=====] - 49s 118ms/step - loss: 0.6222 - accuracy: 0.6509 - val_loss: 0.6402 - val_accu
racy: 0.6298
Epoch 2/30
420/420 [=====] - 48s 115ms/step - loss: 0.6223 - accuracy: 0.6503 - val_loss: 0.6363 - val_accu
racy: 0.6373
Epoch 3/30
420/420 [=====] - 47s 112ms/step - loss: 0.6207 - accuracy: 0.6529 - val_loss: 0.6353 - val_accu
racy: 0.6410
Epoch 4/30
420/420 [=====] - 48s 113ms/step - loss: 0.6193 - accuracy: 0.6540 - val_loss: 0.6374 - val_accu
racy: 0.6380
Epoch 5/30
420/420 [=====] - 45s 107ms/step - loss: 0.6170 - accuracy: 0.6558 - val_loss: 0.6365 - val_accu
racy: 0.6389
Epoch 6/30
420/420 [=====] - 45s 106ms/step - loss: 0.6162 - accuracy: 0.6552 - val_loss: 0.6378 - val_accu
racy: 0.6376
Epoch 7/30
420/420 [=====] - 45s 106ms/step - loss: 0.6142 - accuracy: 0.6592 - val_loss: 0.6370 - val_accu
racy: 0.6423
Epoch 8/30
420/420 [=====] - 44s 105ms/step - loss: 0.6123 - accuracy: 0.6576 - val_loss: 0.6390 - val_accu
racy: 0.6359
Epoch 9/30
420/420 [=====] - 45s 107ms/step - loss: 0.6117 - accuracy: 0.6581 - val_loss: 0.6363 - val_accu
racy: 0.6417
Epoch 10/30
420/420 [=====] - 45s 108ms/step - loss: 0.6092 - accuracy: 0.6623 - val_loss: 0.6387 - val_accu
racy: 0.6374
Epoch 11/30
420/420 [=====] - 44s 105ms/step - loss: 0.6070 - accuracy: 0.6642 - val_loss: 0.6397 - val_accu
racy: 0.6379
Epoch 12/30
420/420 [=====] - 45s 106ms/step - loss: 0.6134 - accuracy: 0.6580 - val_loss: 0.6453 - val_accu
racy: 0.6338
Epoch 13/30
420/420 [=====] - 45s 107ms/step - loss: 0.6076 - accuracy: 0.6612 - val_loss: 0.6431 - val_accu
racy: 0.6365
```

```

Epoch 14/30
420/420 [=====] - 44s 105ms/step - loss: 0.6046 - accuracy: 0.6662 - val_loss: 0.6394 - val_accu
racy: 0.6420
Epoch 15/30
420/420 [=====] - 45s 107ms/step - loss: 0.6020 - accuracy: 0.6683 - val_loss: 0.6368 - val_accu
racy: 0.6386
Epoch 16/30
420/420 [=====] - 45s 106ms/step - loss: 0.5989 - accuracy: 0.6728 - val_loss: 0.6406 - val_accu
racy: 0.6400
Epoch 17/30
420/420 [=====] - 44s 106ms/step - loss: 0.5957 - accuracy: 0.6734 - val_loss: 0.6435 - val_accu
racy: 0.6363
Epoch 18/30
420/420 [=====] - 45s 106ms/step - loss: 0.5946 - accuracy: 0.6736 - val_loss: 0.6406 - val_accu
racy: 0.6356
Epoch 19/30
420/420 [=====] - 45s 107ms/step - loss: 0.5926 - accuracy: 0.6752 - val_loss: 0.6402 - val_accu
racy: 0.6355
Epoch 20/30
420/420 [=====] - 45s 108ms/step - loss: 0.5891 - accuracy: 0.6787 - val_loss: 0.6456 - val_accu
racy: 0.6387
Epoch 21/30
420/420 [=====] - 44s 104ms/step - loss: 0.5865 - accuracy: 0.6822 - val_loss: 0.6461 - val_accu
racy: 0.6385
Epoch 22/30
420/420 [=====] - 46s 108ms/step - loss: 0.5830 - accuracy: 0.6845 - val_loss: 0.6439 - val_accu
racy: 0.6382
Epoch 23/30
420/420 [=====] - 45s 107ms/step - loss: 0.5812 - accuracy: 0.6873 - val_loss: 0.6508 - val_accu
racy: 0.6372
Epoch 24/30
420/420 [=====] - 44s 106ms/step - loss: 0.5796 - accuracy: 0.6876 - val_loss: 0.6505 - val_accu
racy: 0.6407
Epoch 25/30
420/420 [=====] - 44s 106ms/step - loss: 0.5739 - accuracy: 0.6929 - val_loss: 0.6512 - val_accu
racy: 0.6361
Epoch 26/30
420/420 [=====] - 45s 106ms/step - loss: 0.5753 - accuracy: 0.6916 - val_loss: 0.6500 - val_accu
racy: 0.6371
Epoch 27/30
420/420 [=====] - 44s 104ms/step - loss: 0.5705 - accuracy: 0.6943 - val_loss: 0.6538 - val_accu
racy: 0.6303
Epoch 28/30
420/420 [=====] - 45s 107ms/step - loss: 0.5668 - accuracy: 0.6972 - val_loss: 0.6635 - val_accu
racy: 0.6363
Epoch 29/30
420/420 [=====] - 45s 107ms/step - loss: 0.5623 - accuracy: 0.7008 - val_loss: 0.6627 - val_accu
racy: 0.6304
Epoch 30/30
420/420 [=====] - 45s 106ms/step - loss: 0.5608 - accuracy: 0.7017 - val_loss: 0.6596 - val_accu
racy: 0.6333

```

```

In [25]: scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

```

Accuracy: 63.35%

```

In [26]: # Thanks Jason Brownlee
# https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

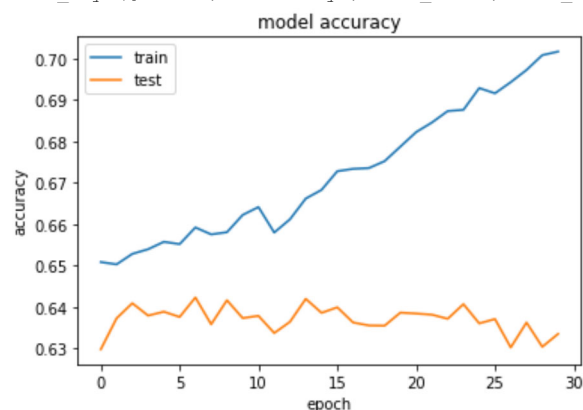
```

```

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





Building the model: 100-50-25 w/ dropout

```
In [31]: model = None

model = Sequential()

model.add(Masking(mask_value=mask_value,
                  input_shape=(series_len, 1)))

model.add(LSTM(100, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(25))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
masking_6 (Masking)	(None, 100, 1)	0
lstm_13 (LSTM)	(None, 100, 100)	40800
dropout_9 (Dropout)	(None, 100, 100)	0
lstm_14 (LSTM)	(None, 100, 50)	30200
dropout_10 (Dropout)	(None, 100, 50)	0
lstm_15 (LSTM)	(None, 25)	7600
dropout_11 (Dropout)	(None, 25)	0
dense_6 (Dense)	(None, 1)	26
Total params: 78,626		
Trainable params: 78,626		
Non-trainable params: 0		
None		

```
In [32]: history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=64)
```

```
Epoch 1/20
420/420 [=====] - 129s 307ms/step - loss: 0.6874 - accuracy: 0.5389 - val_loss: 0.6855 - val_acc
uracy: 0.5412
Epoch 2/20
420/420 [=====] - 124s 295ms/step - loss: 0.6835 - accuracy: 0.5563 - val_loss: 0.6810 - val_acc
uracy: 0.5709
Epoch 3/20
420/420 [=====] - 123s 293ms/step - loss: 0.6746 - accuracy: 0.5739 - val_loss: 0.6670 - val_acc
uracy: 0.5866
Epoch 4/20
420/420 [=====] - 126s 301ms/step - loss: 0.6639 - accuracy: 0.5953 - val_loss: 0.6614 - val_acc
uracy: 0.6039
Epoch 5/20
420/420 [=====] - 125s 297ms/step - loss: 0.6580 - accuracy: 0.6086 - val_loss: 0.6586 - val_acc
uracy: 0.6071
Epoch 6/20
420/420 [=====] - 123s 292ms/step - loss: 0.6545 - accuracy: 0.6121 - val_loss: 0.6587 - val_acc
uracy: 0.6071
Epoch 7/20
420/420 [=====] - 121s 288ms/step - loss: 0.6526 - accuracy: 0.6120 - val_loss: 0.6547 - val_acc
```

```

uracy: 0.6109
Epoch 8/20
420/420 [=====] - 117s 278ms/step - loss: 0.6501 - accuracy: 0.6152 - val_loss: 0.6515 - val_acc
uracy: 0.6110
Epoch 9/20
420/420 [=====] - 121s 288ms/step - loss: 0.6478 - accuracy: 0.6192 - val_loss: 0.6511 - val_acc
uracy: 0.6161
Epoch 10/20
420/420 [=====] - 122s 290ms/step - loss: 0.6457 - accuracy: 0.6225 - val_loss: 0.6481 - val_acc
uracy: 0.6235
Epoch 11/20
420/420 [=====] - 121s 288ms/step - loss: 0.6437 - accuracy: 0.6232 - val_loss: 0.6502 - val_acc
uracy: 0.6188
Epoch 12/20
420/420 [=====] - 122s 290ms/step - loss: 0.6421 - accuracy: 0.6312 - val_loss: 0.6483 - val_acc
uracy: 0.6269
Epoch 13/20
420/420 [=====] - 126s 301ms/step - loss: 0.6386 - accuracy: 0.6355 - val_loss: 0.6414 - val_acc
uracy: 0.6394
Epoch 14/20
420/420 [=====] - 116s 275ms/step - loss: 0.6364 - accuracy: 0.6369 - val_loss: 0.6428 - val_acc
uracy: 0.6320
Epoch 15/20
420/420 [=====] - 115s 275ms/step - loss: 0.6352 - accuracy: 0.6385 - val_loss: 0.6425 - val_acc
uracy: 0.6344
Epoch 16/20
420/420 [=====] - 115s 274ms/step - loss: 0.6351 - accuracy: 0.6397 - val_loss: 0.6402 - val_acc
uracy: 0.6343
Epoch 17/20
420/420 [=====] - 116s 275ms/step - loss: 0.6317 - accuracy: 0.6432 - val_loss: 0.6384 - val_acc
uracy: 0.6341
Epoch 18/20
420/420 [=====] - 116s 276ms/step - loss: 0.6301 - accuracy: 0.6436 - val_loss: 0.6354 - val_acc
uracy: 0.6463
Epoch 19/20
420/420 [=====] - 116s 275ms/step - loss: 0.6281 - accuracy: 0.6473 - val_loss: 0.6390 - val_acc
uracy: 0.6363
Epoch 20/20
420/420 [=====] - 116s 276ms/step - loss: 0.6259 - accuracy: 0.6497 - val_loss: 0.6416 - val_acc
uracy: 0.6376

```

```

In [33]: scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

```

Accuracy: 63.79%

```

In [34]: # Thanks Jason Brownlee
# https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

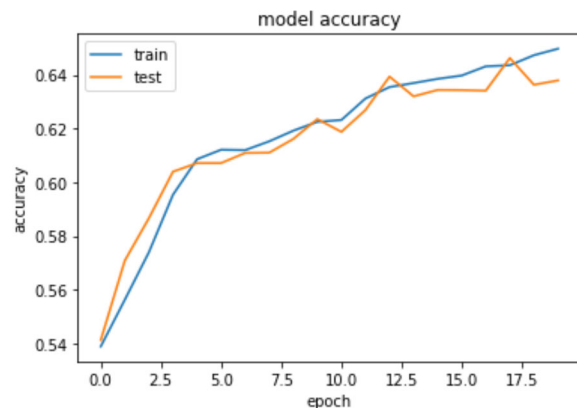
```

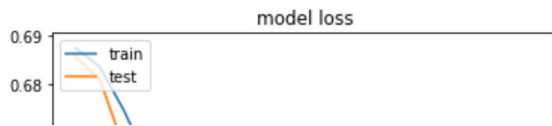
```

print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

```

```
dict keys(['loss', 'accuracy', 'val loss', 'val accuracy'])
```





Building the model: 100-50-25-10 w/ dropout

```
In [51]: model = None

model = Sequential()

model.add(Masking(mask_value=mask_value,
                  input_shape=(series_len, 1)))

model.add(LSTM(100, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(25, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(10))
model.add(Dropout(0.2))

model.add(Dense(1))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

Model: "sequential_16"

Layer (type)	Output Shape	Param #
masking_16 (Masking)	(None, 100, 1)	0
lstm_50 (LSTM)	(None, 100, 100)	40800
dropout_42 (Dropout)	(None, 100, 100)	0
lstm_51 (LSTM)	(None, 100, 50)	30200
dropout_43 (Dropout)	(None, 100, 50)	0
lstm_52 (LSTM)	(None, 100, 25)	7600
dropout_44 (Dropout)	(None, 100, 25)	0
lstm_53 (LSTM)	(None, 10)	1440
dropout_45 (Dropout)	(None, 10)	0
dense_16 (Dense)	(None, 1)	11
dense_17 (Dense)	(None, 1)	2
Total params: 80,053		
Trainable params: 80,053		
Non-trainable params: 0		
None		

```
In [53]: history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=40, batch_size=64)
```

```
Epoch 1/40
420/420 [=====] - 150s 357ms/step - loss: 0.6852 - accuracy: 0.5467 - val_loss: 0.6842 - val_acc
uracy: 0.5423
Epoch 2/40
420/420 [=====] - 146s 349ms/step - loss: 0.6830 - accuracy: 0.5625 - val_loss: 0.6815 - val_acc
uracy: 0.5686
Epoch 3/40
420/420 [=====] - 143s 340ms/step - loss: 0.6804 - accuracy: 0.5671 - val_loss: 0.6780 - val_acc
uracy: 0.5742
Epoch 4/40
420/420 [=====] - 142s 338ms/step - loss: 0.6741 - accuracy: 0.5773 - val_loss: 0.6736 - val_acc
uracy: 0.5746
Epoch 5/40
420/420 [=====] - 141s 336ms/step - loss: 0.6671 - accuracy: 0.5916 - val_loss: 0.6686 - val_acc
uracy: 0.5955
Epoch 6/40
420/420 [=====] - 142s 338ms/step - loss: 0.6602 - accuracy: 0.6056 - val_loss: 0.6603 - val_acc
uracy: 0.6072
Epoch 7/40
420/420 [=====] - 144s 343ms/step - loss: 0.6570 - accuracy: 0.6069 - val_loss: 0.6599 - val_acc
```

```
uracy: 0.6032
Epoch 8/40
420/420 [=====] - 142s 339ms/step - loss: 0.6549 - accuracy: 0.6096 - val_loss: 0.6671 - val_acc
uracy: 0.5863
Epoch 9/40
420/420 [=====] - 145s 345ms/step - loss: 0.6559 - accuracy: 0.6047 - val_loss: 0.6586 - val_acc
uracy: 0.6069
Epoch 10/40
420/420 [=====] - 141s 336ms/step - loss: 0.6528 - accuracy: 0.6123 - val_loss: 0.6527 - val_acc
uracy: 0.6081
Epoch 11/40
420/420 [=====] - 140s 333ms/step - loss: 0.6499 - accuracy: 0.6171 - val_loss: 0.6518 - val_acc
uracy: 0.6111
Epoch 12/40
420/420 [=====] - 142s 338ms/step - loss: 0.6480 - accuracy: 0.6192 - val_loss: 0.6490 - val_acc
uracy: 0.6295
Epoch 13/40
420/420 [=====] - 140s 332ms/step - loss: 0.6457 - accuracy: 0.6228 - val_loss: 0.6469 - val_acc
uracy: 0.6292
Epoch 14/40
420/420 [=====] - 139s 332ms/step - loss: 0.6468 - accuracy: 0.6212 - val_loss: 0.6465 - val_acc
uracy: 0.6239
Epoch 15/40
420/420 [=====] - 138s 329ms/step - loss: 0.6468 - accuracy: 0.6248 - val_loss: 0.6593 - val_acc
uracy: 0.6086
Epoch 16/40
420/420 [=====] - 139s 332ms/step - loss: 0.6440 - accuracy: 0.6244 - val_loss: 0.6443 - val_acc
uracy: 0.6303
Epoch 17/40
420/420 [=====] - 139s 331ms/step - loss: 0.6401 - accuracy: 0.6297 - val_loss: 0.6422 - val_acc
uracy: 0.6364
Epoch 18/40
420/420 [=====] - 139s 331ms/step - loss: 0.6391 - accuracy: 0.6303 - val_loss: 0.6405 - val_acc
uracy: 0.6370
Epoch 19/40
420/420 [=====] - 140s 333ms/step - loss: 0.6378 - accuracy: 0.6336 - val_loss: 0.6435 - val_acc
uracy: 0.6306
Epoch 20/40
420/420 [=====] - 140s 333ms/step - loss: 0.6357 - accuracy: 0.6336 - val_loss: 0.6437 - val_acc
uracy: 0.6241
Epoch 21/40
420/420 [=====] - 139s 331ms/step - loss: 0.6375 - accuracy: 0.6356 - val_loss: 0.6395 - val_acc
uracy: 0.6359
Epoch 22/40
420/420 [=====] - 138s 330ms/step - loss: 0.6347 - accuracy: 0.6351 - val_loss: 0.6391 - val_acc
uracy: 0.6374
Epoch 23/40
420/420 [=====] - 139s 330ms/step - loss: 0.6348 - accuracy: 0.6406 - val_loss: 0.6429 - val_acc
uracy: 0.6341
Epoch 24/40
420/420 [=====] - 138s 329ms/step - loss: 0.6345 - accuracy: 0.6414 - val_loss: 0.6353 - val_acc
uracy: 0.6431
Epoch 25/40
420/420 [=====] - 140s 333ms/step - loss: 0.6304 - accuracy: 0.6417 - val_loss: 0.6383 - val_acc
uracy: 0.6385
Epoch 26/40
420/420 [=====] - 139s 331ms/step - loss: 0.6303 - accuracy: 0.6427 - val_loss: 0.6354 - val_acc
uracy: 0.6389
Epoch 27/40
420/420 [=====] - 139s 331ms/step - loss: 0.6447 - accuracy: 0.6270 - val_loss: 0.6545 - val_acc
uracy: 0.6121
Epoch 28/40
420/420 [=====] - 139s 331ms/step - loss: 0.6550 - accuracy: 0.6113 - val_loss: 0.6479 - val_acc
uracy: 0.6225
Epoch 29/40
420/420 [=====] - 139s 332ms/step - loss: 0.6452 - accuracy: 0.6252 - val_loss: 0.6468 - val_acc
uracy: 0.6216
Epoch 30/40
420/420 [=====] - 139s 330ms/step - loss: 0.6407 - accuracy: 0.6326 - val_loss: 0.6404 - val_acc
uracy: 0.6343
Epoch 31/40
420/420 [=====] - 139s 330ms/step - loss: 0.6360 - accuracy: 0.6354 - val_loss: 0.6381 - val_acc
uracy: 0.6355
Epoch 32/40
420/420 [=====] - 147s 351ms/step - loss: 0.6310 - accuracy: 0.6408 - val_loss: 0.6367 - val_acc
uracy: 0.6403
Epoch 33/40
420/420 [=====] - 140s 333ms/step - loss: 0.6299 - accuracy: 0.6418 - val_loss: 0.6364 - val_acc
uracy: 0.6369
Epoch 34/40
420/420 [=====] - 147s 350ms/step - loss: 0.6296 - accuracy: 0.6434 - val_loss: 0.6334 - val_acc
uracy: 0.6405
Epoch 35/40
420/420 [=====] - 143s 342ms/step - loss: 0.6260 - accuracy: 0.6470 - val_loss: 0.6325 - val_acc
uracy: 0.6402
Epoch 36/40
420/420 [=====] - 149s 354ms/step - loss: 0.6251 - accuracy: 0.6473 - val_loss: 0.6395 - val_acc
uracy: 0.6387
Epoch 37/40
420/420 [=====] - 148s 353ms/step - loss: 0.6271 - accuracy: 0.6464 - val_loss: 0.6318 - val_acc
uracy: 0.6395
Epoch 38/40
420/420 [=====] - 147s 351ms/step - loss: 0.6233 - accuracy: 0.6478 - val_loss: 0.6303 - val_acc
uracy: 0.6459
Epoch 39/40
420/420 [=====] - 138s 328ms/step - loss: 0.6214 - accuracy: 0.6520 - val_loss: 0.6310 - val_acc
uracy: 0.6439
Epoch 40/40
420/420 [=====] - 137s 327ms/step - loss: 0.6205 - accuracy: 0.6540 - val_loss: 0.6296 - val_acc
uracy: 0.6410
```

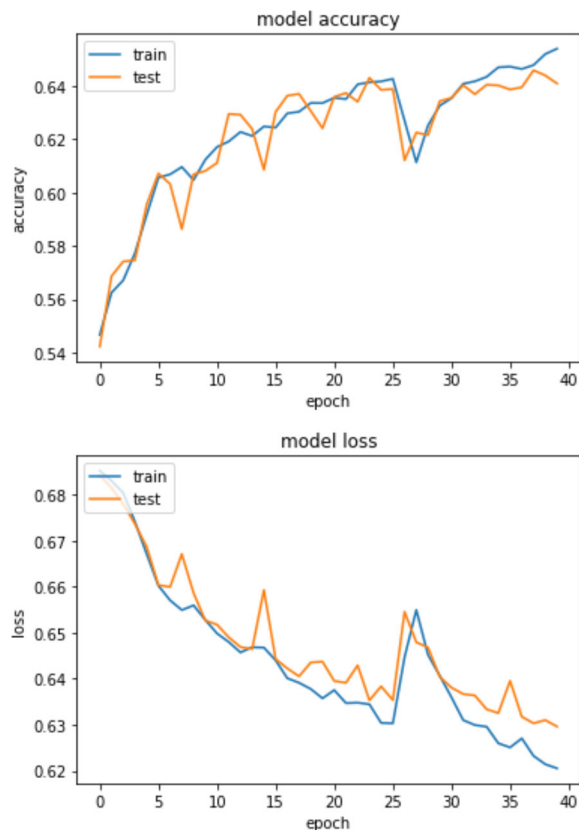
```
In [54]: scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Accuracy: 64.10%

```
In [55]: # Thanks Jason Brownlee
# https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/
```

```
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

dict keys(['loss', 'accuracy', 'val loss', 'val accuracy'])



```
In [57]: if (0):
    model.save(os.path.join('keras', 'LSTM_tree'))
    with open(os.path.join("keras", "LSTM_tree_history.pickle"), 'wb') as f: pickle.dump(history, f) #dump this model
if (0):
    model = keras.models.load_model(os.path.join('keras', 'LSTM_tree'))
```

```
In [ ]: # for layer in model.layers:
#         print(layer.output_shape)

text = "In this example, we developed a working Neural Network for the binary classification problem. The same problem ca
sentiments = text_to_sentiments(text)

print("sentiments shape: ", sentiments.shape)

print("real - predicted")
for num in range(1010,1020):
    scores = bool((model.evaluate(np.array([X_test[num]]), np.array([y_test[num]]), verbose=0, return_dict=True))['accura
prediction = model(np.array([X_test[num]]))
    print("{} = {}: {} - {} ({:.4f})...".format(num, scores, y_test[num], round(float(prediction.numpy())), float(predic
```

```
In [ ]: from keras.utils import plot_model
plot_model(model, to_file='/tmp/model.png', show_shapes=True,)
```

Appendix Lab 06 - Snopes improvements

an abandoned idea

```
In [1]: # imports

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, Masking, Embedding

import random

import os
import pandas as pd
import numpy as np

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sia = SIA()

import nltk
#file for punkt splitter
nltk.download('punkt');
#file for vader sentiment
nltk.download('vader_lexicon');

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

plt.rcParams["figure.figsize"]=20,20
%matplotlib inline
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
```

```
In [2]: series_len = 100
mask_value = -10
max_len = 0

def text_to_sentiments(text):
    sentences = nltk.tokenize.sent_tokenize(text)
    scores = np.vstack([ np.array(list((sia.polarity_scores(s)).values())) for s in sentences]) #list of scores per sentence
    return scores

def rating_to_score_array(t):
    return np.array([int(t=='FALSE'), int(t=='mostly false'), int(t=='mixture'), int(t=='mostly true'), int(t=='TRUE')])
```

```
In [3]: dataset_path = os.path.join("sources", "snopes_checked_v02.csv")
dataset_load = pd.read_csv(dataset_path, encoding='ANSI')

dataset = pd.DataFrame({
    'title': dataset_load['article_title_phase2'],
    'body': dataset_load['original_article_text_phase2'],
    'category': dataset_load['article_category_phase1'],
    'descr_rating': dataset_load['fact_rating_phase1']
})

df = pd.DataFrame(columns = ['scores', 'veracity'])

df['scores'] = dataset.body.apply(text_to_sentiments)
df['veracity'] = dataset.descr_rating.apply(rating_to_score_array)

df.to_csv(os.path.join('out', 'truefake_mcintire.csv'), index=False)

X = np.array(df['scores'].to_list())
X = tf.keras.preprocessing.sequence.pad_sequences(X, maxlen=series_len, padding='post', truncating='pre', dtype='float64')

y = np.array(df['veracity'].to_list())
```

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
X_train.shape
```

```
Out[4]: (234, 100, 4)
```

```
In [5]: model = None

model = Sequential()

model.add(Masking(mask_value=-10,
                  input_shape=(100, 4)))

model.add(LSTM(100, return_sequences=True))

model.add(LSTM(50))

model.add(Dense(5, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())
```

Model: "sequential"

Layer (type)	Output Shape	Param #
masking (Masking)	(None, 100, 4)	0
lstm (LSTM)	(None, 100, 100)	42000
lstm_1 (LSTM)	(None, 50)	30200
dense (Dense)	(None, 1)	51
Total params: 72,251		
Trainable params: 72,251		
Non-trainable params: 0		
None		

```
In [ ]: history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=2)
```

```
In [ ]: # Thanks Jason Brownlee
# https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Appendix Lab 07 - Sentiment & Topics

```
In [2]: # imports
import os
import pandas as pd
import numpy as np

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sia = SIA()

import gensim

import nltk
#file for punkt splitter
nltk.download('punkt');
#file for vader sentiment
nltk.download('vader_lexicon');

#wordnet lemmatization
nltk.download('wordnet')
#more for preprocessing
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *
stemmer = PorterStemmer()

import pyLDAvis.gensim

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"]=20,20
%matplotlib inline

import pickle

import warnings; warnings.simplefilter('ignore')
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

1: Load the dataset

```
In [3]: dataset_path_true = os.path.join("sources", "ISOT", "True.csv")
dataset_path_fake = os.path.join("sources", "ISOT", "Fake.csv")

df_true = pd.read_csv(dataset_path_true, encoding='utf-8') # make sure to use the right encoding
df_fake = pd.read_csv(dataset_path_fake, encoding='utf-8')

dfm_true = df_true.head()
dfm_fake = df_fake.head()

display(dfm_true)
display(dfm_fake)
```

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017
	title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017

2: Split the text into sentences

```
In [4]: def split_sentences(article_text):
        """Takes a string, returns a list of its individual sentences ()"""
        return pd.Series(nltk.tokenize.sent_tokenize(article_text))

sample_sentences = split_sentences(dfm_true.text[0])
display(sample_sentences.iloc[[0,1,-2,-1]])

0    WASHINGTON (Reuters) - The head of a conservat...
1    In keeping with a sharp pivot under way among ...
28   The package far exceeded the $44 billion reque...
29   The Senate has not yet voted on the aid.
dtype: object
```

2.1 Corpus of Sentences

```
In [5]: corpus_sentences = [] # a list of all documents (by sentence) this was intended for sentence level
        #topic detection but never used
```

2.2 Corpus of Entire Texts

```
In [6]: corpus_texts = [] # list of all documents (by entire body)

#corpus_texts = df_true['text'].tolist() + df_fake['text'].tolist()
corpus_texts = df_true['text'].tolist() + df_fake['text'].tolist()

print(len(df_true))
print("amount of documents in corpus: ", len(corpus_texts))

21417
amount of documents in corpus:  44898
```

3 Text Preprocessing

In order to do proper topic analysis the text needs to become understandable by removing uninteresting properties of the text. We lower case it, stem and lemmatize it, and remove all words under 3 characters or stopwords (it them ...).

Now with bi and tri grams!

```
In [7]: def stem_lemmatize(text):
        return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))
```

```
In [8]: def prepare_text(text):
        #all the nice preprocessing without the bigrams and trigrams
        output = []

        for token in gensim.utils.simple_preprocess(text):
            if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:
                output.append(stem_lemmatize(token))
        return output

sample_text = "At the time of her abdication at age 75, Beatrix was the oldest reigning monarch in the country's history."
prep_sample_text = prepare_text(sample_text)
```

```
In [11]: %%time

prepared_corpus_texts = [prepare_text(d) for d in corpus_texts]
print(prepared_corpus_texts[7])

['follow', 'statement', 'post', 'verifi', 'twitter', 'account', 'presid', 'donald', 'trump', 'realdonaldtrump', 'potu', '
opinion', 'express', 'reuter', 'edit', 'statement', 'confirm', 'accuraci', 'realdonaldtrump', 'fake', 'news', 'love', 'ta
lk', 'call', 'approv', 'rat', 'foxandfriend', 'show', 'rat', 'approxim', 'presid', 'obama', 'despit', 'massiv', 'neg', 't
rump', 'coverag', 'russia', 'hoax', 'unit', 'state', 'post', 'offic', 'lose', 'billion', 'dollar', 'year', 'charg', 'amaz
on', 'littl', 'deliv', 'packag', 'make', 'amazon', 'richer', 'post', 'offic', 'dumber', 'poorer', 'charg', 'sourc', 'link
', 'jpexyr']
Wall time: 0 ns
```

```
In [15]: %%time

bigram = gensim.models.phrases.Phrases(prepared_corpus_texts, min_count=5, threshold=20) # 5 and 10 are the default values
trigram = gensim.models.phrases.Phrases(bigram[prepared_corpus_texts], threshold=20)

bigram_mod = gensim.models.phrases.Phruaser(bigram)
trigram_mod = gensim.models.phrases.Phruaser(trigram)

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

Wall time: 2min 4s
```

```
In [19]: # print(type(bigram))

# bigram_freeze = bigram.freeze()
# trigram_freeze = trigram.freeze()
bigram.save(os.path.join('gensim', 'bigram.pkl'))
trigram.save(os.path.join('gensim', 'trigram.pkl'))
```

```
In [25]: print(make_trigrams([prepared_corpus_texts[2]]))
print("\n\n")
print(corpus_texts[2])
print("\n\n")
print(trigram_mod[bigram_mod[preprocess("donald trump")]])
```

```
[[('washington reuter', 'special_counsel investig', 'link', 'russia', 'presid', 'trump', 'elect', 'campaign', 'continu', 'interfer', 'despit', 'call', 'trump', 'administr', 'alli', 'republican', 'lawmak', 'shut', 'promin', 'republican', 'senat', 'say', 'sunday', 'lindsey_graham', 'serv', 'senat', 'arm_forc', 'judiciari_committe', 'say', 'depart_justic_special_counsel', 'robert_mueller', 'need', 'carri', 'russia', 'investig', 'polit', 'interfer', 'investig', 'forward', 'investig', 'conduct', 'polit', 'influenc', 'graham', 'say', 'face', 'nation', 'news', 'program', 'need', 'mueller', 'think', 'right', 'right', 'time', 'question', 'russia_interfer_elect', 'trump', 'campaign', 'link', 'ordin', 'effort', 'loom', 'white_hous', 'trump', 'take', 'offic', 'januari', 'show', 'sign', 'reced', 'trump', 'prepar', 'second', 'year', 'power', 'despit', 'intensifi', 'rhetor', 'trump', 'alli', 'recent', 'week', 'accus', 'mueller_team', 'bia', 'republican', 'presid', 'trump', 'undercut', 'support', 'interview', 'week', 'york_time', 'say', 'expect', 'mueller', 'go', 'fair', 'russia', 'role', 'elect', 'question', 'possibl', 'link', 'trump', 'campaign', 'focu', 'multipl', 'inquiri', 'washington', 'committe', 'senat', 'hous_repres', 'investig', 'mueller_team', 'take', 'earlier', 'probe', 'launch', 'feder_bureau_investig', 'member', 'trump', 'campaign', 'administr', 'convict', 'indict', 'investig', 'trump', 'alli', 'deni_collus_russia', 'campaign', 'kremlin_deni_meddl_elect', 'graham', 'say', 'want', 'examin', 'dossier', 'link', 'trump', 'russia', 'compil', 'british_christoph_steel', 'prompt', 'trump', 'alli', 'republican', 'question', 'mueller', 'inquiri', 'saturday', 'york_time', 'report', 'dossier', 'trigger', 'earli', 'probe', 'trump', 'campaign', 'foreign_polic_i_advis', 'georg_papadopoulos', 'australian', 'diplomat', 'russia', 'damag', 'inform', 'trump', 'rival_hillari_clinton', 'want', 'somebod_i', 'look', 'depart_justic', 'dossier', 'bother', 'greatli', 'want', 'somebod_i', 'look', 'graham', 'say', 'say', 'russia', 'investig', 'continu', 'matter', 'fact', 'hurt', 'ignor', 'say']]]
```

WASHINGTON (Reuters) - The special counsel investigation of links between Russia and President Trump's 2016 election campaign should continue without interference in 2018, despite calls from some Trump administration allies and Republican lawmakers to shut it down, a prominent Republican senator said on Sunday. Lindsey Graham, who serves on the Senate armed forces and judiciary committees, said Department of Justice Special Counsel Robert Mueller needs to carry on with his Russia investigation without political interference. "This investigation will go forward. It will be an investigation conducted without political influence," Graham said on CBS's Face the Nation news program. "And we all need to let Mr. Mueller do his job. I think he's the right guy at the right time." The question of how Russia may have interfered in the election, and how Trump's campaign may have had links with or co-ordinated any such effort, has loomed over the White House since Trump took office in January. It shows no sign of receding as Trump prepares for his second year in power, despite intensified rhetoric from some Trump allies in recent weeks accusing Mueller's team of bias against the Republican president. Trump himself seemed to undercut his supporters in an interview last week with the New York Times in which he said he expected Mueller was "going to be fair." Russia's role in the election and the question of possible links to the Trump campaign are the focus of multiple inquiries in Washington. Three committees of the Senate and the House of Representatives are investigating, as well as Mueller, whose team in May took over an earlier probe launched by the U.S. Federal Bureau of Investigation (FBI). Several members of the Trump campaign and administration have been convicted or indicted in the investigation. Trump and his allies deny any collusion with Russia during the campaign, and the Kremlin has denied meddling in the election. Graham said he still wants an examination of the FBI's use of a dossier on links between Trump and Russia that was compiled by a former British spy, Christopher Steele, which prompted Trump allies and some Republicans to question Mueller's inquiry. On Saturday, the New York Times reported that it was not that dossier that triggered an early FBI probe, but a tip from former Trump campaign foreign policy adviser George Papadopoulos to an Australian diplomat that Russia had damaging information about former Trump rival Hillary Clinton. "I want somebody to look at the way the Department of Justice used this dossier. It bothers me greatly the way they used it, and I want somebody to look at it," Graham said. But he said the Russia investigation must continue. "As a matter of fact, it would hurt us if we ignored it," he said.

```
[('donald_trump']
```

```
In [21]: def preprocess(text):
         return(trigram_mod[bigram_mod[prepare_text(text)]])
```

4 Dictionary Creation

word embeddings, all occurring words are stored and get a number (embedding) those embeddings can later be used for vector calculations. of course not all words are important, so words that appear more than 100000 times or that exist in >60% our corpus (those are very likely words slipped the stopword list) and less than 15 time (words not important enough for a topic)

```
In [28]: %%time

# Creating the gensim dictionary of word embeddings

if (0):
    dictionary = gensim.corpora.Dictionary.load(os.path.join("gensim", "dictionary"))

if (1):

    processed_corpus_texts = [trigram_mod[bigram_mod[doc]] for doc in prepared_corpus_texts]

    dictionary = gensim.corpora.Dictionary(processed_corpus_texts)
    print("unfiltered: ", dictionary)

    dictionary.filter_extremes(no_below=15, no_above=0.6, keep_n=100000)

print(" filtered: ", dictionary)
```

```
unfiltered: Dictionary(116439 unique tokens: ['administr', 'agre', 'aid', 'approach', 'approv']...)
```

```
filtered: Dictionary(21592 unique tokens: ['administr', 'agre', 'aid', 'approach', 'approv']...)
```

```
In [29]: dictionary.save(os.path.join("gensim", "dictionary"))
```

5 Bag of Words creation

Now we create a vector representation in the form of a bag of words for each document. a vector that lets us know how often each word in the preprocessed text, that also exists in our dictionary, occurs

```
In [30]: %%time

if (0):
    with open(os.path.join('gensim', 'bag_of_words.pickle'), 'rb') as f: bow_corpus_texts = pickle.load(f)

if (1):
    bow_corpus_texts = [dictionary.doc2bow(text) for text in processed_corpus_texts]

dictionary.doc2bow(prepare_sample_text)
```

Wall time: 8.04 s

```
Out[30]: [(30, 1), (86, 1), (329, 1), (5940, 1), (6606, 1), (9052, 1), (9059, 1)]
```

6 LDA model creation

```
In [60]: %%time

num_topics = 15
if (0):
    lda_model = gensim.models.LdaMulticore.load(os.path.join("gensim", "lda_model20"))
if (0):
    lda_modelX = gensim.models.LdaMulticore(corpus = bow_corpus_texts,
                                             id2word = dictionary,
                                             num_topics = num_topics,
                                             passes = 20,
                                             iterations = 200,
                                             eta = 'auto',
                                             #alpha = 'auto',
                                             workers = 2)

    # multicore speeds up the process significantly
```

Wall time: 9min 7s

7 TF-IDF x LDA model creation

```
In [31]: # train the model
tfidf_model = gensim.models.TfidfModel(bow_corpus_texts)
tfidf_corpus_texts = tfidf_model[bow_corpus_texts]
```

```
In [32]: tfidf_model.save(os.path.join('gensim', 'tfidf_model.pkl'))
```

```
In [76]: %%time

num_topics = 15

if (0):
    lda_tfidf_model = gensim.models.LdaMulticore(corpus = tfidf_corpus_texts,
                                                  id2word = dictionary,
                                                  num_topics = num_topics,
                                                  passes = 20,
                                                  iterations = 200,
                                                  eta = 'auto',
                                                  #alpha = 'auto', # not available form multicore
                                                  workers = 2)

if (0):
    lda_tfidf_model.save(os.path.join("gensim", "02", "lda_model")) #save model itself
    with open(os.path.join("gensim", "02", "tfidf_corpus.pickle"), 'wb') as f: pickle.dump(tfidf_corpus_texts, f) #dump
    dictionary.save(os.path.join("gensim", "02", "dictionary")) #save dictionary

    # multicore speeds up the process significantly
```

Wall time: 2.64 s

```
In [67]: %time

from gensim.models import CoherenceModel# Compute Coherence Score

coherence_model_lda = CoherenceModel(model=lda_modelX, texts=processed_corpus_texts, dictionary=dictionary, coherence='c
coherence_lda = coherence_model_lda.get_coherence()

coherence_model_lda_tfidf = CoherenceModel(model=lda_tfidf_model, texts=processed_corpus_texts, dictionary=dictionary, co
coherence_lda_tfidf = coherence_model_lda_tfidf.get_coherence()

print('\nCoherence Score: ', coherence_lda)
print('\nCoherence Score: ', coherence_lda_tfidf)

Coherence Score: 0.47852099608521953

Coherence Score: 0.5345937527916345
Wall time: 1min 39s
```

8 Case Examination

take a look at some classifications

```
In [ ]: #TODO LDAVIS For TWEAKING SHIT SAVE PICTURES AND STEPS WHOOO

pyLDAvis.enable_notebook()
visualisation = pyLDAvis.gensim.prepare(lda_model50, bow_corpus_texts, dictionary)
pyLDAvis.save_html(visualisation, os.path.join("gensim", "LDAvis", "LDAvis.html"))

In [86]: article = 30000

print("\nmost frequently occurring word: ", max(preprocess(corpus_texts[article]),key=preprocess(corpus_texts[article]).co

print("\n\nbow-lda prediction:")

for index, score in sorted(lda_modelX[bow_corpus_texts[article]],
                           key=lambda tup: -1*tup[1]):
    print("\nScore: {} \t \nTopic: {}".format(score, lda_modelX.print_topic(index, 10)))

print("\n\nntfidf-lda prediction:")

for index, score in sorted(lda_tfidf_model[tfidf_corpus_texts[article]],
                           key=lambda tup: -1*tup[1]):
    print("\nScore: {} \t \nTopic: {}".format(score, lda_tfidf_model.print_topic(index, 10)))

print("\n\n", preprocess(corpus_texts[article]), "\n\n", corpus_texts[article])

most frequently occurring word: trump

bow-lda prediction:

Score: 0.5868434309959412
Topic: 0.014*"like" + 0.013*"peopl" + 0.011*"know" + 0.011*"go" + 0.010*"think" + 0.010*"obama" + 0.009*"presid" + 0.009
*"american" + 0.009*"trump" + 0.008*"time"

Score: 0.29268181324005127
Topic: 0.161*"trump" + 0.039*"presid" + 0.026*"clinton" + 0.024*"donald_trump" + 0.017*"campaign" + 0.016*"white_hous" +
0.010*"report" + 0.008*"hillari_clinton" + 0.008*"http" + 0.008*"tell"

Score: 0.09060100466012955
Topic: 0.012*"year" + 0.011*"republican" + 0.009*"million" + 0.009*"plan" + 0.009*"fund" + 0.008*"congress" + 0.007*"hou
s" + 0.007*"american" + 0.007*"state" + 0.007*"govern"

Score: 0.022879285737872124
Topic: 0.015*"women" + 0.014*"black" + 0.011*"protest" + 0.010*"student" + 0.010*"white" + 0.009*"support" + 0.008*"schoo
l" + 0.008*"video" + 0.007*"twitter" + 0.007*"peopl"

tfidf-lda prediction:

Score: 0.8023303151130676
Topic: 0.005*"trump" + 0.003*"obama" + 0.002*"american" + 0.002*"peopl" + 0.002*"clinton" + 0.002*"presid" + 0.002*"like"
+ 0.002*"elect" + 0.002*"hillari" + 0.002*"parti"

Score: 0.0666574165225029
Topic: 0.009*"wire" + 0.007*"cuba" + 0.007*"cuban" + 0.007*"assang" + 0.006*"berni_sander" + 0.005*"berni" + 0.005*"white
_helmet" + 0.005*"broadcast_live" + 0.004*"meme" + 0.004*"julian_assang"

Score: 0.02224014885723591
Topic: 0.032*"deep_state" + 0.021*"hanniti" + 0.007*"gorka" + 0.007*"undercov" + 0.007*"oligarch" + 0.006*"russian_collu
s" + 0.006*"kimmel" + 0.006*"khan" + 0.006*"pakistan" + 0.005*"hernandez"

Score: 0.020426075905561447
Topic: 0.009*"moor" + 0.008*"reilli" + 0.008*"conway" + 0.007*"funni" + 0.005*"bannon" + 0.005*"hammond" + 0.005*"savag"
+ 0.004*"kellyann_conway" + 0.004*"warren" + 0.004*"kushner"
```

```
['hous', 'card', 'star', 'kevin_spacey', 'play', 'fiction_charact', 'presid', 'frank_underwood', 'cnbc', 'intern', 'thur
sday', 'couldn', 'avoid', 'get', 'ask', 'question', 'concern', 'presidenti_elect', 'chief', 'question', 'happen', 'donald
_trump', 'debat', 'frank_underwood', 'answer', 'give', 'draw_laughter', 'wouldn', 'debat', 'underwood', 'terribl', 'accid
t', 'debat', 'terribl', 'wasn', 'funni', 'follow', 'best', 'trump', 'imperson', 'huuug', 'huge', 'church', 'spacey', 'curr
ent', 'make', 'round', 'globe', 'promot', 'upcom', 'season', 'hous', 'card', 'air', 'march', 'ask', 'question', 'annual',
```

```
'skybridg_capit', 'recept', 'host', 'piano', 'earlier', 'say', 'get', 'ask', 'question', 'happen', 'frank_underwood', 'debat', 'trump', 'think', 'say', 'rememb', 'import', 'distinct', 'charact', 'fiction_charact', 'fiction_charact', 'turn', 'line', 'past', 'probabl', 'best', 'elabor', 'answer', 'provid', 'session', 'septemb', 'think', 'underwood', 'kill', 'donald_trump', 'trump', 'elect', 'think', 'great', 'donald_trump', 'start', 'feud', 'fiction_charact', 'mayb', 'fair', 'fight', 'trump', 'appear', 'season', 'hous', 'card', 'year', 'lose', 'real', 'elect', 'novemb', 'person', 'fetch', 'trump', 'rival', 'frank_underwood', 'featur_imag_screen_captur']
```

House of Cards star Kevin Spacey, who plays the fictional character President Frank Underwood, sat down with CNBC International on Thursday and just couldn't avoid getting asked questions concerning the 2016 presidential election. Chief among them was a question about what would happen if Donald Trump ever had to debate Frank Underwood. Here is the answer he gave, which drew quite a bit of laughter: He wouldn't (debate Underwood). There would be a terrible accident, on the way to the debate. It would be terrible, and very sad. If that wasn't funny enough, he followed that up with his best Trump impersonation: I'm huuuuge. I'm too huge for this church. Spacey is currently making rounds across the globe promoting the new upcoming season of House of Cards, which airs in March. He was also asked this very same question at the annual SkyBridge Capital reception hosted at a piano bar just a day earlier. He says he often gets asked the question of what would happen if Frank Underwood had to debate Trump, who would win? I thought about this and I said, 'We must remember one important distinction, one of these characters is a fictional character, and the other is a fictional character.' It turns out he's used this line in the past often. Probably the best and most elaborate answer he's ever provided is from a Q & A session with CNN he did back in September. Oh, I think Underwood (would win). He (would win) because he would kill Donald Trump. Trump would never make it to election day. It'd be over, done. I think it would be great if Donald Trump started a feud with a fictional character, because maybe for once it would be a fair fight. Perhaps Trump will appear in season 5 of House of Cards next year after he loses the real U.S. election this November. He is, after all, a TV personality. It's not too

X Saving items

because we don't want to sit waiting every time please

```
In [16]: if (0):
         lda_modelXX.save(os.path.join("gensim", "lda_modelXX"))
         with open(os.path.join("gensim", "bag_of_words.pickle"), 'wb') as f: pickle.dump(bow_corpus_texts, f)
         dictionary.save(os.path.join("gensim", "dictionary"))
```

10: Performing sentiment analysis

Which we can do either to the whole article or on a sentence basis and then average

```
In [71]: def get_scores(text: list, method='VADER'):
         if method == 'VADER':
             scores = text.apply(lambda s: sia.polarity_scores(s)['compound']) #list of compound score per sentence
         else:
             scores = None

         return scores
```

```
In [72]: def average_sentiment(doc):
         doc_sentence_list = split_sentences(doc)
         return float(np.average(get_scores(doc_sentence_list)))

         average_sentiment(corpus_texts[10005])
```

```
Out[72]: 0.10535714285714284
```

as can be seen in the following cell the entire document sentiment gives crazy extreme and unnuanced values

```
In [74]: article = 10005

         sample_article = corpus_texts[article]
         sample_sentences = split_sentences(sample_article)

         sample_scoreT = sia.polarity_scores(sample_article)['compound']
         sample_scoreS = get_scores(sample_sentences)

         print(sample_scoreT, "\n\n", average_sentiment(sample_article), "\n\nfrom:\n", sample_scoreS)

0.9153

0.10535714285714284

from:
0      0.4588
1     -0.3818
2     -0.3818
3      0.4019
4      0.0000
5      0.0000
6      0.1779
7      0.1531
8      0.8442
9      0.4215
10     -0.2960
11      0.0000
12      0.0772
13      0.0000
dtype: float64
```

```
In [105... %%time

topic_cols = ['Sentiment']+(['T'+str(x)) for x in range(0, num_topics)]+['Veracity']
# text to list of topic scores

def docid_to_row(doc_id):
    new_row = dict.fromkeys(topic_cols , 0)

    new_row['Sentiment'] = average_sentiment(corpus_texts[doc_id])
    for index, score in lda_tfidf_model[tfidf_corpus_texts[doc_id]]:
        new_row['T'+str(index)] = score
    if(doc_id<21417):
        new_row['Veracity'] = 1
    return(new_row)

print(docid_to_row(0))

{'Sentiment': 0.07614666666666665, 'T0': 0, 'T1': 0, 'T2': 0, 'T3': 0, 'T4': 0, 'T5': 0, 'T6': 0, 'T7': 0, 'T8': 0, 'T9': 0, 'T10': 0.026416631, 'T11': 0, 'T12': 0, 'T13': 0.8555344, 'T14': 0.054770425, 'Veracity': 1}
Wall time: 17 ms
```

```
In [106... %%time

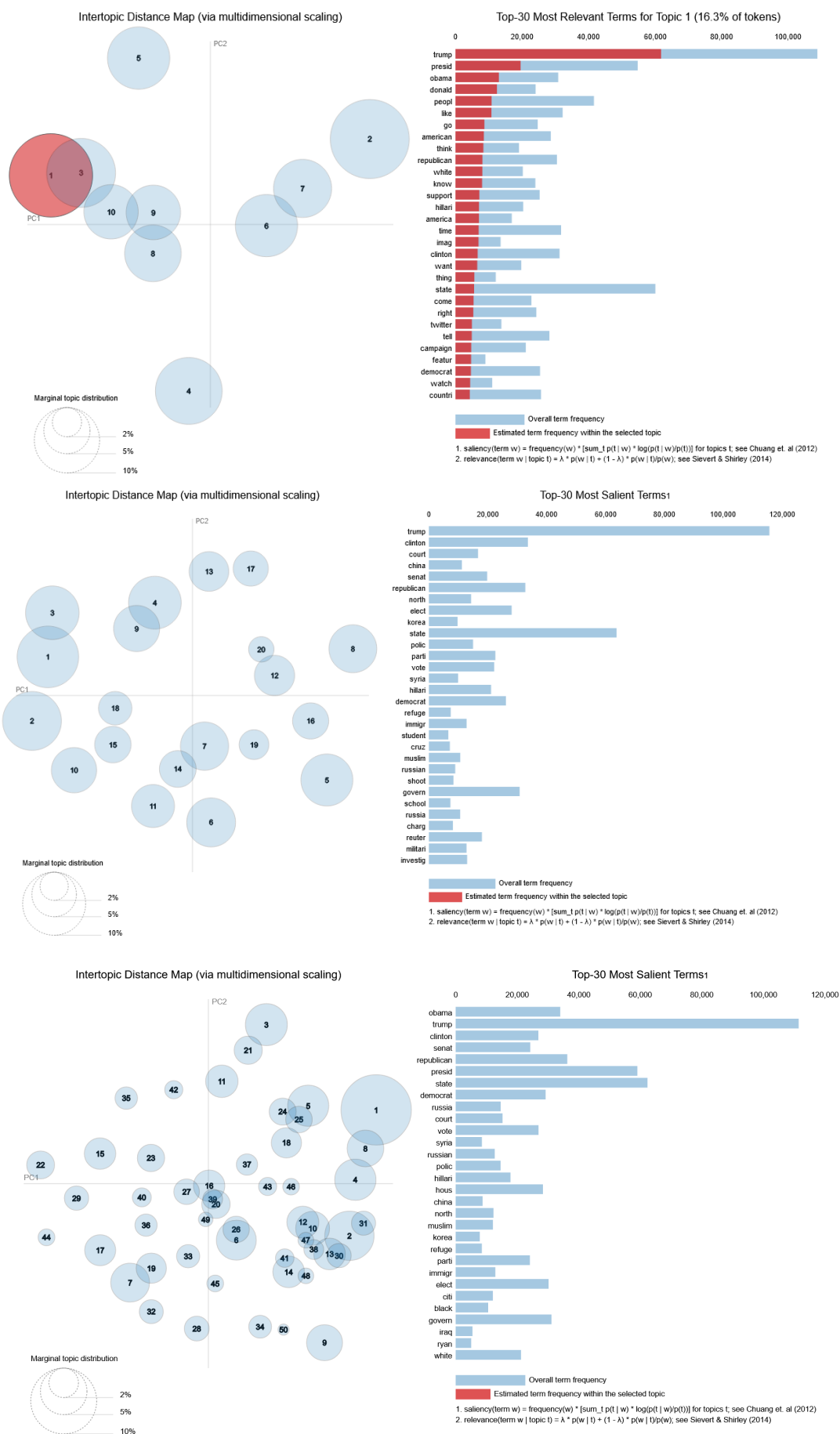
df_topic_sentiment = pd.DataFrame(columns=topic_cols)

for doc_id in range(len(corpus_texts)):
    df_topic_sentiment = df_topic_sentiment.append(docid_to_row(doc_id), ignore_index=True)

Wall time: 7min 5s
```

```
In [108... df_topic_sentiment.to_csv(os.path.join('out', 'topic_sentiment.csv'), index=False)
```


I. Appendix - PyLDAvis



Appendix Lab 08 - Model combination

combining the sentiment sequence LSTM and the topic LDA with statistics

```
In [80]: # imports
import os
import pandas as pd
import numpy as np

import sys

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sia = SIA()

import gensim

import nltk
#file for punkt splitter
nltk.download('punkt');
#file for vader sentiment
nltk.download('vader_lexicon');

#wordnet lemmatization
nltk.download('wordnet')
#more for preprocessing
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()

import pyLDAvis.gensim

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"]=20,20
%matplotlib inline

import pickle

import warnings; warnings.simplefilter('ignore')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]   C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]   C:\Users\emiel\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
In [22]: import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, Masking, Embedding
```

1: Load the dataset

```
In [9]: dataset_path_true = os.path.join("sources", "ISOT", "True.csv")
dataset_path_fake = os.path.join("sources", "ISOT", "Fake.csv")

df_true = pd.read_csv(dataset_path_true, encoding='utf-8') # make sure to use the right encoding
df_fake = pd.read_csv(dataset_path_fake, encoding='utf-8')

dfm_true = df_true.head()
dfm_fake = df_fake.head()

display(dfm_true)
display(dfm_fake)
```

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017
	title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn't wish all Americans ...	News	December 31, 2017

	title	text	subject	date
1	Drunk Bragging Trump Staffer Started Russian	House Intelligence Committee Chairman Devin Nu	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke	On Friday, it was revealed that former Milwauk	News	December 30, 2017
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017

```
In [10]: corpus_texts = [] # list of all documents (by entire body)

#corpus_texts = df_true['text'].tolist() + df_fake['text'].tolist()
corpus_texts = df_true['text'].tolist() + df_fake['text'].tolist()

print("amount of documents in corpus: ", len(corpus_texts))

amount of documents in corpus: 44898
```

X: Creating the new Data

Loading models

```
In [15]: with open(os.path.join('gensim', '02', 'tfidf_corpus.pickle'), 'rb') as f: tfidf_corpus_texts = pickle.load(f)

lda_tfidf_model = gensim.models.LdaMulticore.load(os.path.join("gensim", "02", "lda_model"))
```

```
In [33]: # model = keras.models.load_model(os.path.join('keras', 'LSTM_tree'))

pre_activation_layer_name = 'dense_16'

model_pre_activation = keras.Model(inputs=model.input,
                                   outputs=model.get_layer(pre_activation_layer_name).output)

model_pre_activation.summary()
```

Model: "functional_3"

Layer (type)	Output Shape	Param #
masking_16_input (InputLayer	[(None, 100, 1)]	0
masking_16 (Masking)	(None, 100, 1)	0
lstm_50 (LSTM)	(None, 100, 100)	40800
dropout_42 (Dropout)	(None, 100, 100)	0
lstm_51 (LSTM)	(None, 100, 50)	30200
dropout_43 (Dropout)	(None, 100, 50)	0
lstm_52 (LSTM)	(None, 100, 25)	7600
dropout_44 (Dropout)	(None, 100, 25)	0
lstm_53 (LSTM)	(None, 10)	1440
dropout_45 (Dropout)	(None, 10)	0
dense_16 (Dense)	(None, 1)	11
Total params: 80,051		
Trainable params: 80,051		
Non-trainable params: 0		

```
In [50]: series_len = 100
mask_value = -10
max_len = 0

def text_to_sequence(text):
    # feed corpus_texts[entry], returns 100 units long -10 padded sequence of sentiment data
    sentences = nltk.tokenize.sent_tokenize(text)
    scores = [sia.polarity_scores(s)['compound'] for s in sentences] #list of compound score per sentence

    if len(scores) < series_len:
        t = series_len - len(scores)
        scores = np.pad(scores, (t, 0), mode='constant', constant_values=mask_value)
        scores = scores.reshape((series_len, 1))
    else:
        scores = np.array(scores[(-1*series_len):])
        scores = scores.reshape((series_len, 1))

    return scores

def scores_to_sequence(scores):
    if len(scores) < series_len:
        t = series_len - len(scores)
        scores = np.pad(scores, (t, 0), mode='constant', constant_values=mask_value)
        scores = scores.reshape((series_len, 1))
    else:
        scores = np.array(scores[(-1*series_len):])
        scores = scores.reshape((series_len, 1))
    return scores

sample_sentences = split_sentences(corpus_texts[2])
sample_scores = get_scores(sample_sentences)
sample_sequence = scores_to_sequence(sample_scores)
sample_prediction = float(model(np.array([sample_sequence])))

print(sample_prediction)

0.617904543876648
```

```
In [53]: %%time

num_topics = 15

def split_sentences(article_text):
    """Takes a string, returns a list of its individual sentences ()"""
    return pd.Series(nltk.tokenize.sent_tokenize(article_text))

def get_scores(text: list, method='VADER'):
    if method == 'VADER':
        scores = text.apply(lambda s: sia.polarity_scores(s)['compound']) #list of compound score per sentence
    else:
        scores = None

    return scores

def sentiment_stats(doc):
    doc_sentence_list = split_sentences(doc)
    sentiments = get_scores(doc_sentence_list)
    sequence = scores_to_sequence(sentiments)
    output = {'LSTM_pred' : float(model(np.array([sequence])))}
    output.update(sentiments.describe().to_dict())
    return (output)

sentiment_stats(corpus_texts[2])

Wall time: 354 ms
```

```
Out[53]: {'LSTM_pred': 0.617904543876648,
'count': 17.0,
'mean': -0.01017647058823531,
'std': 0.45047517208139753,
'min': -0.6808,
'25%': -0.2732,
'50%': 0.0,
'75%': 0.3818,
'max': 0.7269}
```

functions for operation

```
In [54]: %%time

stat_cols = ['count', 'mean', 'std', 'min', '25%', '50%', '75%', 'max', 'LSTM_pred']
topic_cols = [('T'+str(x)) for x in range(0, num_topics)]+['Veracity']

# text to list of topic scores

def docid_to_row(doc_id):
    new_row = dict.fromkeys(topic_cols , 0)

    new_row.update(sentiment_stats(corpus_texts[doc_id])) # add the sentiment key and value pairs

    for index, score in lda_tfidf_model[tfidf_corpus_texts[doc_id]]: # update the topic distributions
        new_row['T'+str(index)] = score

    if(doc_id<21417): # the first 21417 items in the DB are
        new_row['Veracity'] = 1

    return(new_row)

print(docid_to_row(2))

{'T0': 0, 'T1': 0, 'T2': 0, 'T3': 0, 'T4': 0, 'T5': 0.54721564, 'T6': 0, 'T7': 0, 'T8': 0, 'T9': 0, 'T10': 0, 'T11': 0, 'T12': 0, 'T13': 0.3566922, 'T14': 0, 'Veracity': 1, 'LSTM_pred': 0.617904543876648, 'count': 17.0, 'mean': -0.01017647058823531, 'std': 0.45047517208139753, 'min': -0.6808, '25%': -0.2732, '50%': 0.0, '75%': 0.3818, 'max': 0.7269}
Wall time: 502 ms
```

```
In [81]: %%time

df_topic_sentiment = None
df_topic_sentiment = pd.DataFrame(columns=stat_cols+topic_cols)

count_texts = len(corpus_texts)
for doc_id in range(count_texts):
    df_topic_sentiment = df_topic_sentiment.append(docid_to_row(doc_id), ignore_index=True)
    sys.stdout.write('\r')
    sys.stdout.write(str("current progress: {}".format(str(np.round(doc_id/count_texts*100, 3)))))

sys.stdout.write('\r')
sys.stdout.write('building table done!')
display(df_topic_sentiment.head())

building table done!.998%

count    mean    std    min    25%    50%    75%    max    LSTM_pred    T0  ...  T6    T7  T8  T9    T10  T11  T12    T13
0      30.0    0.076147  0.402857 -0.6369 -0.077025 0.0000 0.31820 0.9062    0.830285 0.000000 ... 0.0 0.000000 0.0 0.0 0.026417 0.0 0.0 0.855568 0.0
1      21.0    0.018648  0.486143 -0.8625 -0.292400 0.0772 0.34000 0.7531    0.331971 0.000000 ... 0.0 0.000000 0.0 0.0 0.000000 0.0 0.0 0.776423 0.0
2      17.0   -0.010176  0.450475 -0.6808 -0.273200 0.0000 0.38180 0.7269    0.617905 0.047824 ... 0.0 0.000000 0.0 0.0 0.000000 0.0 0.0 0.342261 0.0
3      16.0   -0.029581  0.438841 -0.6249 -0.382650 0.0000 0.35045 0.7430    0.571730 0.179278 ... 0.0 0.000000 0.0 0.0 0.000000 0.0 0.0 0.280279 0.0
4      40.0    0.047153  0.327357 -0.6124 -0.156025 0.0000 0.26705 0.6249    0.785731 0.000000 ... 0.0 0.051978 0.0 0.0 0.000000 0.0 0.0 0.842692 0.0

5 rows x 25 columns

Wall time: 4h 22min 32s

In [82]: df_topic_sentiment.to_csv(os.path.join('out','everything.csv'), index=False)
```