Slovak University of Technology in Bratislava
Faculty of Electrical Engineering and Information Technology

Registration number: FEI-5384-92272

Automatic pre-selection of potential misinformation for later expert evaluation

Master's thesis

2022 **Bc. Jakub Fedorko**

Slovak University of Technology in Bratislava
Faculty of Electrical Engineering and Information Technology

Registration number: FEI-5384-92272

Automatic pre-selection of potential misinformation for later expert evaluation

Master's thesis

Study Programme:	Applied Informatics
Study Field:	Computer Science
Training Workplace:	Kempelen Institute of Intelligent Technologies and Institute of Computer Science and Mathematics
Supervisor:	doc. Ing. Jakub Šimko, PhD.

Bratislava 2022 **Bc. Jakub Fedorko**

Slovak University of Technology in Bratislava
Institute of Computer Science and Mathematics

Faculty of Electrical Engineering and Information Technology
Academic year: 2021/2022
Reg. No.: FEI-5384-92272

STU FEI

MASTER THESIS TOPIC

Student: **Bc. Jakub Fedorko**
Student's ID: 92272
Study programme: Applied Informatics
Study field: Computer Science
Thesis supervisor: doc. Ing. Jakub Šimko, PhD.
Head of department: Dr. rer. nat. Martin Drozda
Consultant: doc. Ing. Jakub Šimko, PhD.
Workplace: KInIT and ÚIM FEI STU

Topic: **Automatic pre-selection of potential misinformation for later expert evaluation**

Language of thesis: English

Specification of Assignment:

Journalists and activists try to mitigate the spread of false information via fact-checking: systematic verification of content (i.e. news articles, audiovisual records, social media contributions) factual veracity. Fact-checking is a cognitively demanding task which still has to be performed by humans. However, it can be supported by automatic approaches such as automatic pre-selection of check-worthy content (it reduces the amount of content a human has to review) or identification of features indicating mendacity or sources that speed up fact-checking (i.e. sources of relevant and verified information). The automatic methods can be based on various approaches ranging from natural language analysis to context and source behavioural analysis.

Tasks:
1. Analyze existing approaches to automatic fact-checking support and automatic detection of false information.
2. Design and experimentally verify the method for automatic fact-checking support, which conducts pre-selection of check-worthy content (that is, the content with an increased probability of false information occurrence). The method can also assist human fact-checkers by providing suitable indicators of false information or referencing credible sources of information.
3. Identify or acquire a suitable data sample, which will be used for experimental verification of the proposed method.

Recommended literature:
Cheema, G. S., Hakimov, S., & Ewerth, R. (2020). Check_square at checkthat! 2020: Claim detection in social media via fusion of transformer and syntactic features. arXiv preprint arXiv:2007.10534.

Hansen, C., Hansen, C., Simonsen, J. G., & Lioma, C. (2019, September). Neural Weakly Supervised Fact Check-Worthiness Detection with Contrastive Sampling-Based Ranking Loss. In CLEF (Working Notes).

Zeng, X., Abumansour, A. S., & Zubiaga, A. (2021). Automated fact - checking: A survey. Language and Linguistics Compass, 15(10), e12438.

Deadline for submission of Master thesis: 13. 05. 2022
Approval of assignment of Master thesis: 16. 05. 2022
Assignment of Master thesis approved by: prof. RNDr. Gabriel Juhás, PhD. – Study programme supervisor

SÚHRN

Slovenská technická univerzita v Bratislave
Fakulta elektrotechniky a informatiky

Študijný program:	Aplikovaná informatika
Autor:	Bc. Jakub Fedorko
Diplomová práca:	Automatický predvýber potenciálne nepravdivých správ pre neskoršie posúdenie expertom
Vedúci záverečnej práce::	doc. Ing. Jakub Šimko, PhD.
Miesto a rok predloženia práce:	Bratislava 2022

Manuálne metódy overovania faktov sa stávajú neúčinnými v dôsledku rapídneho šírenia nepravdivých informácií prostredníctvom sociálnych sietí. Automatické systémy overovania faktov sa pokúšajú zmierniť potenciálne škody spôsobované nepravdivými informáciami detekciou overovania hodných tvrdení, zhromažďovaním relevantných informácií k týmto tvrdeniam a vyvodzovaním pravdivosti týchto tvrdení. Práce súčasného výskumu v oblasti vhodnosti overovania sa pri sprostredkovaní významu textu v čoraz väčšom rozsahu spoliehajú na technológie slovných a vetných embeddingov. Nahrádzajú tým jednoduchšie textové črty ako tf-idf, sentiment a iné, nájdené predovšetkým v starších prácach. Motivovaní vyššie uvedenými skutočnosťami a tiež všeobecným nedostatkom využitia technológie vetných embeddingov, predstavujeme náš `sent-nn` model. Ten reprezentuje text vetnými embeddingami založenými na BERT jazykovom modeli. Náš model prekonal obe metódy určené na porovnanie o viac ako 2.5 % a 4.2 % v metrikách average precision a f1-score na pozitívnej triede.

Kľúčové slová: fact-checking, check-worthiness, sentence transformers

ABSTRACT

Slovak University of Technology in Bratislava
Faculty of Electrical Engineering and Information Technology

Study Programme:	Applied Informatics
Author:	Bc. Jakub Fedorko
Master's thesis:	Automatic pre-selection of potential misinformation for later expert evaluation
Supervisor:	doc. Ing. Jakub Šimko, PhD.
Place and year of submission:	Bratislava 2022

The rapid spread of false information on social media renders manual fact-checking methods ineffective. Automatic fact-checking systems try to inhibit potential damage caused by false information by detecting check-worthy claims, gathering relevant information, and inferring their veracity. The recent research in the check-worthiness field relies increasingly on word and even sentence embeddings technology to convey meaning, substituting simpler text features such as tf-idf, sentiment and others found in earlier works. Motivated by the above and general scarcity of use of the sentence embeddings, we present our `sent-nn` model utilising BERT-based sentence embeddings as its text representation, which outperforms baseline methods by more than 2.5% and 4.2% in average precision and f1-score on the positive class.

Keywords: fact-checking, check-worthiness, sentence transformers

Acknowledgements

I want to thank my supervisor, doc. Ing. Jakub Šimko, PhD., for professional mentoring and expert advice. Furthermore, I want to express my gratitude to Swiss Re Management AG in Bratislava and, specifically Ing. Lukáš Csóka, for allowing me to use their GPU cluster for training.

Contents

Chapter 1

Introduction

On social media, false information spreads faster and to more users than legitimate information, as shown by Vosoughi et al. [45]. Combined with the fact that more than 60% of people acquire their news on social media[1], this creates a genuine threat to human society. Organizations like FactCheck.org, PolitiFact or Demagog try to mitigate the potential damage of false information by manually and comprehensively fact-checking societally relevant claims present in public space. However, the lengthy and demanding process of manual fact-checking cannot keep up with the easy unrestricted creation and spread of false information.

Therefore, the need for an automatic fact-checking system arises, although it proves to be just as complex a problem as its manual equivalent. An automatic fact-checking pipeline usually consists of three main steps: detecting check-worthy parts of the text (i.e., sentences containing the claim that should be fact-checked), collecting information relevant to the check-worthy claim, and finally, using this information to infer the veracity of the check-worthy claim. Most research in this area primarily focused on the last two steps of the pipeline. These steps, collecting relevant information and inferring the veracity of check-worthy claims, usually expect check-worthy claims as their input. However, the step of acquiring check-worthy claims was, until recently, much less researched. That has changed with the inception of the CheckThat! Lab[2], a competition by the CLEF association devoted to finding a state of the art solutions to natural language processing (NLP) tasks such as claim retrieval, fake news detection and check-worthiness estimation.

The earlier approaches rely on relatively simple tf-idf features [21, 33] to represent their words and transitively sentences, whereas the more recent works represent sentences with much more sophisticated and meaningful word [46, 17, 7, 29] or even sentence [24] embeddings. Approaches extend their sentence representations with text features such as sentence length [21, 33, 46], sentiment [21, 33, 46], part-of-speech (POS) tags [21, 33, 46, 24, 7], named entities (NE) [21, 33, 46, 24] and syntactic dependencies [46, 17] trying to convey in them as much meaning as possible. Although

[1] `https://pewrsr.ch/3My8ZE9`

[2] `https://sites.google.com/view/clef2021-checkthat/home`

as embeddings technology improves, the value of text features diminishes, and thus the later approaches tend to simplify their sentence representations in terms of text features while retaining the meaning via embeddings. The use of state of the art embeddings models seems to be the most impactful decision one can make when designing a check-worthiness solution.

The goal of this work is two-fold:

1. *find, modify if need be and run available check-worthiness solutions to act as baseline methods,*
2. *design and implement an original solution to outperform said baseline methods.*

Inspired by the great works in the check-worthiness area of research and motivated by the fact that no prior work has utilized sentence embeddings based on the Bidirectional Encoder Representations from Transformers (BERT)[3] language model, we train two separate models, both employing BERT based embeddings. The first model, inspired by Hansen et al. [17], is centred around a bidirectional LSTM layer and makes use of word embeddings extracted with the BERT base model. The second one is a simple neural network that takes inspiration from Konstantinovskiy et al.'s use of sentence embeddings [24]. We experiment with our word and sentence representation by extracting features based on less and more granular POS tags and syntactic dependencies. For training, highly unbalanced data (see Figure 3.1) comprised of transcripts of multiple political debates provided by the CLEF association for 2019 CheckThat! Lab [1] is used. To mitigate the issues of class imbalance and general scarcity of the data, we experiment with training on a considerably larger domain-specific dataset weakly labelled with the existing check-worthiness solution.

To acquire baseline methods for comparison with our models, we download and modify two check-worthiness solutions [7, 29] to run with the data we use. We evaluate our models with average precision and f1-score on the positive class to conduct a fair comparison as baseline methods optimized the former metric and our models the latter. Our best performing model, which uses sentence embeddings as its sentence representation, achieves 4.36% and 2.62% performance improvements in average precision over Cheema et al.'s [7] and Martinez et al.'s [29] solutions, respectively. It also outperforms both baseline methods in f1-score by 4.24% and 13.77%.

Our contribution is a neural network classifier implemented in *PyTorch*[4], which utilizes a pre-trained sentence transformer to encode sentences into sentence embeddings.

[3] `https://huggingface.co/docs/transformers/model_doc/bert`
[4] `https://pytorch.org/`

Chapter 2

Problem analysis

2.1 Journalistic fact-checking

Fact-checking as a concept originates in New York City in the 1920s. TIME magazine[1] founders then hired new employees to ensure everything gathered by the reporters was accurate [39]. Today fact-checking is considered an essential part of journalistic work. Shapiro et al. [38] have defined journalism as a "discipline of verification" to discriminate it from "entertainment, propaganda, fiction or art". Fact-checking and verification are often used interchangeably, even though Silverman [39] sees verification as a "discipline that lies at the heart of journalism, and that is increasingly being practiced and applied by other professions" and fact-checking as a "specific application of verification in the world of journalism". Furthermore, Mantzarlis [28] says fact-checking "addresses the claim's logic, coherence, and context", while Kovach and Rosenstiel [25] define verification as a "scientific-like approach of getting the fact and also the right facts", which often involves reviewing the source, date, and location of materials. We could say verification is the necessary step in the process of fact-checking.

A new, but related meaning for fact-checking emerged in 2003 with the launch of FactCheck.org[2], a website whose goal is to "monitor the factual accuracy of what is said by major U.S. political players in the form of TV ads, debates, speeches, interviews and news releases." [39] Later, it was joined by PolitiFact[3], Demagog[4] and many others. According to Duke Reporters' Lab [40], number of fact-checking organizations more than doubled between years 2016 and 2020, reaching over 300 total. Fact-checking organizations like these usually rate statements on a scale based on how truthful the statements are, e.g. Politifact's trademarked scale *Truth-O-Meter* runs from "True" to "Pants on Fire." Graves [16] describes the actual process of fact-checking as consisting of five distinct areas of practise. The first and vital function of fact-checker is to choose claims to check. Once the fact-checker has chosen a claim to check, they contact the author

[1] `https://time.com/`
[2] `https://www.factcheck.org/`
[3] `https://www.politifact.com/`
[4] `https://demagog.sk/`

of the claim. This practise serves as a matter of fairness toward the author as well as an investigative technique. Whether the contacting of the author was successful or not, fact-checker continues by tracing claim's origin and reconstructing its spread in hopes of acquiring important contextual cues. Next comes verification during which fact-checker must rely on official data from government agencies and often needs to consult experts. Final practise to present a verdict on the scale, followed by review of presented verdict.

Both individual practises and the process in its entirety are non-trivial and require substantial amount of time. In case of PolitiFact, this amount of time can be estimated ranging from one to three days per fact-check.

2.2 The spread of misinformation

Misinformation is information that may not be accurate or complete [23]. We can also say it is false information. Vosoughi et al. [45] investigated differences in the spread of true and false information on Twitter between 2006 and 2017. Their data consisted of 126,000 stories tweeted more than 4.5 million times. They classified these stories based on an agreement between 6 independent fact-checking organizations. They found that "falsehood diffused significantly farther, faster, deeper, and more broadly than the truth in all categories of information." False stories were retweeted more times than true stories - the fraction of false stories which were retweeted ten or more times was significantly greater than true stories. They reached more people - the top 1% of false stories regularly reached between 1,000 and 100,000 people, while true stories rarely reached 1,000 people. Furthermore, false information reached 1,500 people six times faster than truth.

To put this information into perspective, Leskovec et al. [27], tracking 1.6 million mainstream media sites and blogs over three months to observe the news cycle's dynamics, found a typical lag between peaks of attention to a phrase in the news media and blogs is 2.5 hours.

Even though Leskovec et al. do not discriminate between true and false information and their type of information slightly differ from Vosoughi's Twitter stories, it is reasonable to assume Twitter stories travel at approximately the same speed, if not faster.

The previous two sections lay a firm ground that supports the need for an automated fact-checking solution as they contrast the lengthy and thorough process of manual fact-checking with the speed and depth of the spread of false information

2.3 Automated fact-checking

A typical automated fact-checking pipeline consists of three crucial steps: choosing check-worthy sentences, obtaining information related to those sentences, and finally inferring the veracity of the check-worthy sentences based on related information [17]. However, specific implementations of such pipeline differ. Thorne and Vlachos [42] divide many existing approaches by inputs, sources of evidence, and outputs.

Input-wise, they describe three categories of approaches used: triplets (subject - predicate - object) [31, 9, 4], textual claims [29, 7], and even entire documents [44, 20, 17, 33, 46].

By sources of evidence, they mean types of evidence used for fact-checking. There are five different types of sources described: no evidence besides the claim itself [36], knowledge graphs [9, 41, 44], pure text (such as encyclopedia articles, policy documents, verified news, and scientific journals) [14], repositories of previously checked claims [21], and finally, aggregate information on the distribution of posts on social media networks [11]. Different types of evidence have a different impact on the model as well as the output of the model.

As for outputs, they describe simple binary true and false [31], a scale of values ranging between true and false (similar to one described in section 2.1) [36], triplets scored within a numerical range indicating how likely they are to be accurate [3], article's stance towards the claim (supported, refuted, neutral, irrelevant) [14], output consisting of the label (supported, refuted, not enough information) and the sentences required to reach the verdict if the label states supported or refuted [43].

It is important to note that although these approaches all share the same goal, their definitions of tasks being automated vary and consequently vary their inputs, sources of evidence, and outputs.

2.4 Fact-check worthiness estimation approaches

2.4.1 ClaimBuster

The development on ClaimBuster has started in December 2014 and lasts to the present time. However, this analysis covers its state as described in [20, 21]. ClaimBuster is considered to be the first end-to-end automatic fact-checking system. It is composed of three main components: claim monitor, claim spotter, and claim matcher. The claim monitor monitors various sources of information to obtain data for the claim spotter. Claim spotter identifies check-worthy sentences containing factual claims. Claim matcher finds closely related or identical fact-checked claims. There is also a component called a claim checker, which queries external knowledge bases, and the Web, but only in case that claim matcher could not find any similarities. The following text focuses on the functionality of the claim spotter, described in detail in [21].

Hassan et al. constructed their own dataset from U.S. presidential debates between 1960 and 2012. There were a total of 15 presidential elections in that time, out of which 12 had debates before the event. After reducing the number of sentences from the collected corpus by removing those not spoken by a candidate and those shorter than five words, they ended up with 20,788 sentences. They also developed a ground-truth collection website for dataset annotation purposes. A monetary reward system was put in place to encourage high-quality labelling. The final class distribution was 66.31% of NFS, 10.17% of UFS, and 23.52% CFS (labels are described in detail in a paragraph dedicated to the classification task).

ClaimBuster's check-worthiness pipeline takes a sentence as its input. We will see that almost all other analysed approaches chose sentence as theirs' model input as well. Features described in the following paragraph are then extracted from the sentence, forming a new sentence representation. These new sentence representations are used as training data for supervised methods employed by ClaimBuster team. In the end, the check-worthiness pipeline returns a score in the range from 0.0 to 1.0 for each sentence indicating how likely a given sentence contains a factual claim.

The following features categories were extracted from sentences: sentiment, length, part-of-speech (POS) tags and entity type. To calculate sentiment features, they used AlchemyAPI[5], which produced a sentiment score ranging from -1 for the most negative sentiment to 1 for the most positive sentiment. The length feature category represented the word count of a tokenised sentence. They used Natural Language Toolkit (NLTK)[6] for the sentence tokenisation as well as POS tagging. There were 43 POS tags found in the corpus, each having its own feature defined as the number of words in a sentence belonging to a given POS tag. Similarly to the sentiment feature category, entity types were also extracted using AlchemyAPI. A total of 2,727 entities from labelled sentences belonged to 26 distinct types. The entity type feature corresponded to the number of entities of a given type in a sentence. Hassan et al. performed feature selection by training a random forest classifier to find the best discriminating features. GINI index was used to determine features' importance in constructing each decision tree. They calculated the overall importance of a feature as its average importance over all the tress. They found that the most discriminating features are POS tags referring to past tense (VBD) and cardinal numbers (CD).

The authors of ClaimBuster refer to the check-worthiness task as claim spotting. They modelled the claim spotting task to be partly classification task and partly ranking task. In the classification task, they categorised sentences into three categories: Non-Factual Sentences (NFS), Unimportant Factual Sentences (UFS), Check-worthy Factual Sentences (CFS). Subjective sentences containing opinions or beliefs and many questions belonged to the NFS category, i.e. "But I think it's time to talk about the future." Sentences with factual claims which were not check-worthy were part of the UFS category, i.e. "Two days ago we ate lunch at a restaurant." Finally, sentences that contained check-worthy factual claims were included in the CFS category, i.e. "Over a million and a quarter Americans are HIV-positive." They used three supervised learning methods, namely Multinomial Naive Bayes Classifier (NBC), Support Vector Machine (SVM) and Random Forest Classifier (RFC), to conduct this classification task. Methods were evaluated utilising 4-fold cross-validation with SVM reaching the best precision and recall across all three classes. They also experimented with multiple combinations of extracted features. Models' performance was tested on word features (tf-idf) alone, then on word features together with POS tags, and finally word, POS tags and entity type features. While entity type features improved the performance of NBC, for the other two models, these features either did not affect performance or were detrimental. POS tags features

[5] `https://pypi.org/project/AlchemyAPI/`

[6] `https://www.nltk.org/`

were much more helpful, however for the best scoring SVM model, they only accounted for a 0.2% increase in precision and a 0.1% increase in recall (both precision and recall were weighted averages across all three classes).

The ranking task was defined as deriving "a score that reflects the degree by which a sentence belongs to CFS." They achieved this by treating sentences from both NFS and UFC categories as negatives and only CFS sentences as positives. They then employed SVM to find a decision boundary between the two classes. Lastly, they followed Platt's scaling technique [35] to calculate a posterior probability of the sentence belonging to CFS using SVM's decision function. Sentences were ranked according to their probabilities. This SVM model reached an average precision of 97.9% and 89.7% on the top 100 and 500 sentences, respectively.

2.4.2 TATHYA

Patwari et al. [33] suggest that a set of check-worthy text is a subset of the more extensive set described as checkable text and that check-worthiness is not consistent across statements with similar content. Their multi-classifier system TATHYA utilises this information by identifying latent groupings of data that best describe whether a statement is check-worthy or not.

Similarly to the ClaimBuster team, the authors of TATHYA created their own dataset from political debates. However, they chose primary and presidential debates from 2016 and included Donald Trump's Presidential Announcement Speech to analyse a single person discourse. A total of 21,700 collected statements was reduced to 15,735 after removing all of the statements with less than two tokens. It is important to note that tokens were extracted after the removal of stop-words and frequently used words. They discovered a significant class imbalance with only 967 statements labelled as check-worthy. The process of labelling relied on fact-checking results from multiple fact-checking organisations like Factcheck.org, Politifact, Washington Post, and others. If any of the fact-checking organisations fact-checked a sentence, TATHYA team labelled that sentence as check-worthy. During empirical analysis, they found that there are inconsistencies in the form of small overlaps on checked statements between fact-checking organisations. Consequently, they asked two human annotators to find more check-worthy sentences. Human annotators agreed on 145 additional check-worthy statements.

TATHYA's fact-checking unit is a single sentence, adding to the previously outlined pattern. The multi-classifier system accepts its feature representation and returns either check-worthy or not. TATHYA is one of two analysed approaches that return discrete binary values.

Patwari et al. build on top of ClaimBuster's features but include topic of discussion, entity history, POS tuples, and frequently occurring phrases. To acquire discussion topic features, they train an LDA topic model [6] with Gibbs sampling[7] on the dataset and tune the number of topics to 30. They then calculate the topic probability distribution and define context size x. For $x/2$ previous and following sentences, they calculate co-

[7] `https://pypi.python.org/pypi/lda`

sine similarity. The authors introduce new entity history features. Each sentence has its entity history of size h, which holds all the entities appearing in the previous h sentences. If any entity within the entity history of a sentence is repeated in that sentence, they activate one of two features (*entity_type*, *discuss*), (*entity_type*, *repeat*) based on the speaker. The POS tuples features are built on the proposition that claims often have a subject, verb, object dependency structure. They focus on subject and verb to capture references of self and opponent. An example of such POS tuple is (*noun_tag*, *verb_tag*, *neg*) with the corresponding sentence "She did not." The approaches by Hansen et al. [17] and Zuo et al. [46] would later replace this feature with the features based on dependency parse trees. They use Stanford CoreNLP[8] and NLTK[9] for tokenisation, POS-tagging, NER-tagging and Coreference-Resolution.

Patwari et al. justify the use of a multi-classifier system by claiming that "Multi-classifier systems have been shown to improve performance when a single classifier system lacks expressiveness." [33] This multi-classifier is trained by following a training algorithm that starts with clustering data into the same number k of groups as there are classifiers. As their classifiers, they use SVMs with linear kernel from the scikit-learn[10] module, and clustering is done using the Kmeans algorithm. In the successive steps, they iteratively train classifiers on the groups and evaluate their ability to predict the check-worthiness of the training sentences. Hyperparameters tuning is done via grid search on cross-validation. The best groupings are then found based on the classifier confidence in their prediction. They found that their multi-classifier system achieves the best results on the test set when the number of groups k equals 3. Patwari et al. performed a comparison between the ClaimBuster and TATHYA on the test set comprising of only presidential and vice-presidential debates. TATHYA out-performed ClaimBuster in recall and f1-score but achieved a lower precision score. Their scores were 0.188 and 0.226 precision, 0.248 and 0.148 recall, and 0.214 and 0.179 f1-score, respectively.

It is important to note that the authors performed an ablation study where they trained a single SVM classifier on various combinations of sentence feature representations. They discovered that adding POS tags and entity types improves the model by 4%.

2.4.3 Zuo et al.

The work by Zuo et al. [46] was a winning submission to 2018 CheckThat! Lab [32]. The definition of their task was:

> Predict which claim in a political debate should be prioritized for fact-checking. In particular, given a debate, the goal is to produce a ranked list of its sentences based on their worthiness for fact-checking [32].

[8] `https://stanfordnlp.github.io/CoreNLP`

[9] `http://www.nltk.org`

[10] `http://scikit-learn.org/stable/modules/svm.html`

They were provided with English and Arabic versions of the dataset however, they chose to only work with the English version due to their intention of using heuristics that rely on linguistic insight.

The English version of the dataset included three political debates as training data and two political debates and five speeches as test data. It is important to note that the distinction between debates and speeches in the test set was made by Zuo et al., not authors of 2018 CheckThat! Lab. Training data consisted of extremely imbalanced 3,989 sentences, out of which only 94 were labelled as check-worthy.

The authors extract a vast number of features, including those described in previous works. However, prior to the feature extraction, they process the data by normalizing speaker names to unify different names referring to the same person and extracting all the sentences by a speaker to create sub-datasets in the training data that represent speech-like texts. The latter is done to help train the model as political speeches are included in the test dataset. Then the feature extraction begins with removing stopwords and stemming the remaining terms with Snowball stemmer[11]. On top of POS tags and sentence length, they add a number of tokens in past, present and future tenses inferred from POS tags and a number of negations in a sentence as features. They create dependency parse trees containing clause and phrase-level tags. A number of words within the scope of each tag is extracted as a feature. There are also affective features such as subjectivity, direct and associated bias, and opinion. Metadata features that describe whether the speaker's opponent is mentioned, the speaker is the moderator, or whether a strong reaction follows a sentence are included. However, many fact-checking organisations are reluctant to use systems where metadata features regarding the speaker are utilised [24]. They define a segment as a maximal set of consecutive sentences by the same speaker. The use of segments yields the following features: relative position of a sentence within its segment and sentence count in previous, current, and subsequent segments. This work is the first of the analysed works to utilise word embeddings, which are a great tool to capture contextual information. They use 300-dimensional Google News word embeddings[12] to represent each word as a vector. The arithmetic mean is then taken of the vectors corresponding to words in a sentence to get an abstract sentence embedding used as a feature. To battle the high dimensionality of their feature space, they employ two distinct dimensionality reduction methods. First, they apply a feature selection module from scikit-learn[13] library to perform univariate feature selection with x^2-test to select the best 2,000 features. Second, they train SVM with linear kernel and L1 regularization, motivated by the observation that linear models with L1 regularization encourage the vanishing coefficients for weakly correlated features. After applying these methods, the total number of features is reduced to 2,655 for debates and 2,404 for speeches.

Zuo et al. train SVM, MLP and an ensemble of both models for the check-worthiness ranking task. Both models utilise L2 regularization to combat over-fitting. The MLP

[11] `http://snowball.tartarus.org/texts/introduction.html`

[12] `https://code.google.com/archive/p/word2vec/`

[13] `https://scikit-learn.org/stable/index.html`

model has two layers with 100 and 8 neurons, a hyperbolic tangent as an activation function and adam as its optimizer. The ensemble model's score consists of scores by SVM and MLP, normalised and averaged. An adaptive synthetic sampling algorithm for imbalanced learning, ADASYN, is used to overcome significant class imbalance during training. They use 3-fold cross-validation for model selection and a 2-fold version for speeches. As previously mentioned, the authors only work with the English dataset to be able to use heuristics. These heuristics require linguistic understanding to be formulated and address the following properties of sentences: whether the speaker of a sentence is a candidate, the length of a sentence, whether the sentence contains the phrase "thank you," the number of subjects in the sentence, and whether the sentence ends with a question mark. There are thresholds regarding these properties, based on which heuristic rules override scores assigned by the classification models. The primary evaluation metric is MAP. Their best scoring model is MLP with strict heuristics with 0.1366 MAP.

2.4.4 Konstantinovskiy et al.

Konstantinovskiy et al. [24] are the first to develop a claim detection system that leverages universal sentence representations. They also introduce the first annotation schema for claim detection, developed by experts at Full Fact[14].

The authors created their own dataset via the crowdsourcing annotation method. However, prior to creating the dataset, they initiated the development of the first annotation schema for claim detection. The annotation schema, developed by experts at Full Fact, consisted of 7 different categories: personal experience, quantity in the past or present, correlation or causation, current laws or rules of operation, prediction, another type of claim, and not a claim. They recruited 80 volunteers to annotate 6,304 sentences extracted from subtitles of four UK political TV shows. Due to the differences between the annotations from volunteers, they had to apply an agreement strategy. The agreement strategy where at least 60% of the participants had to agree on the annotation would eliminate too many sentences. Therefore they chose the majority vote strategy where at least three annotators marked the sentence, and at least half of the annotators agreed. This strategy selected 4,080 sentences for previously outlined seven categories and 4,777 sentences for two categories - claim and not claim.

Konstantinovskiy et al. extract considerably fewer features compared to previous approaches. Sentence embeddings, POS tags and named entities are their only features. However, they compensate for this lack of features with more sophisticated embeddings. They use InferSent encoder published and later publicly released by Conneau et al. [10]. The InferSent embeddings differ from word embeddings by utilising a recurrent neural network to account for word order. The InferSent method first converts the words to their common crawl GloVe[15] representations, then passes them through bidirectional long-short-term memory (BiLSTM) network [22]. They pre-train these

[14] `https://fullfact.org/`

[15] `https://nlp.stanford.edu/projects/glove/`

sentence embeddings on a large corpus of Natural Language Inference tasks[16]. POS tags and named entities are included in the form of a feature vector concatenated to the sentence embedding. The feature vector contains the count of each POS or NE tag in a sentence. However, experiments conducted by the authors reveal that their most successful classifier does not benefit from having access to POS or NE information within an embedding.

The authors train Logistic Regression, Linear SVM, Gaussian Naive Bayes, and Random Forest supervised classifiers from the scikit-learn library, all with default parameters. They compare classifiers performance-wise among each other as well as among baseline approaches of other teams in the aforementioned experiments. Their Logistic Regression classifier gives the highest overall f1-score of 0.83, outperforming the best ClaimRank-based model by 6% and best ClaimBuster-based model by 4%.

2.4.5 Hansen et al.

Similarly to Zuo et al. [46], the approach by Hansen et al. [17] is also a winning submission to CheckThat! Lab, although in the following year of 2019. The subsequent analysis, however, is based on their article published in Companion Proceedings of The 2019 World Wide Web Conference. Hansen et al. present a neural check-worthiness approach, which utilises various novel ideas such as dual word representations fed to the recurrent neural network or weak supervision.

As for data, they use three datasets, each with a different purpose. The Embedding Training Dataset is comprised of all US election-related documents available through American Presidency Project[17]. It contains 15,059 documents, and they use it to pre-train domain-specific word embeddings. The Evaluation Dataset consists of 2,602 sentences from seven check-worthiness annotated political speeches from the 2016 US elections. Finally, the Weakly Labelled Dataset contains all publicly available speeches by Hillary Clinton and Donald Trump from the 2016 US elections. They weakly label a total of 37,732 sentences using the public API of ClaimBuster[18].

Each word in the Hansen et al. approach is represented by its word embedding and syntactic dependencies. There are no other categories of features extracted. The motivation behind dual word representation lies within the understanding that the word embedding seeks to represent the semantics of the word in its context, while the syntactic dependencies of a word strive to identify the role of that word in modifying the semantics of other words in the sentence. Furthermore, they highlight Le et al. [26] finding that the top-weighted words in check-worthy and non-check-worthy sentences overlap. They hypothesise that the syntactic dependencies of a word may help distinguish these overlapping top-weighted words. Their domain-specific pre-trained embeddings are based on the word2vec[19] skip-gram model. The syntactic dependencies of words are

[16] `https://nlp.stanford.edu/projects/snli/`

[17] `https://web.archive.org/web/20170606011755/http://www.presidency.ucsb.edu/`

[18] `https://idir.uta.edu/claimbuster/api/`

[19] `https://code.google.com/archive/p/word2vec/`

parsed using the spaCy syntactic parser. Section 5.2 mentioned above analyses the effects of word embeddings and syntactic dependencies on the model's performance. The domain-specific pre-trained embeddings in combination with syntactic dependencies score the best with 0.302 MAP. Syntactic dependencies account for 1.7% improvement as the domain-specific pre-trained embeddings without syntactic dependencies score 0.285 MAP.

The dual word representations are fed to a recurrent neural network (RNN) with GRU [8] memory units. They aggregate the output of each word in RNN using an attention mechanism. A fully connected layer then accepts the attention-weighted sum to predict an output using the sigmoid activation function. They use RMSprop optimizer and binary cross-entropy as the loss function. They tune and evaluate using 7-fold cross-validation. The neuron ratio between GRU cell and single fully connected layer is 4 to 1. They provide a comparison of the effectiveness of multiple check-worthiness approaches, both with and without the use of weak supervision. Their model outperforms the second-best approach without weak supervision by Konstantinovskiy et al. by 1.1% with 0.278 MAP and the second-best approach with supervision by Gencheva et al. by 6.6% with 0.302 MAP. However, it is crucial to note that theirs is the only one of the models to benefit from using weak supervision.

2.4.6 Cheema et al.

Cheema et al. [7] participated in 2020 CheckThat! Lab [2], where they submitted their approaches to tasks 1 - Tweet Check-Worthiness English and Arabic, and 2 - Claim Retrieval. However, only the English version of task 1 is subject to the following analysis. Along with the submission, they released the code for their approaches to the public.

They use an English version of the dataset provided by CheckThat! Lab with 962 tweets on the topic of Covid-19. The tweets are divided into training, development, and test splits with 672, 150 and 140 tweets, respectively.

Cheema et al. tackle the problem of a small dataset by extracting various features to acquire a rich feature representation. With a plethora of elements like hashtags and user mentions, the tweets require excessive pre-processing. They use the Baziotis et al. [5] tool to apply tokenization, lower-casing, removal of punctuation, spell correction, and normalization of hashtags, all-caps, censored, elongated and repeated words and elements like URL, email, phone number or user mentions. They use spaCy[20] to extract POS, NE and syntactic dependencies. There are 16 POS tags extracted in total and later reduced to eight based on their importance during empirical evaluation. For the features based on syntactic dependencies, they use relations between tokens within a given tweet. The dependency relation is used if POS tags of parent and child nodes belong to a subset containing ADJ, ADV, NOUN, PROPN, VERB or NUM. Such dependency relations are then converted into pairs (*child node-POS, parent node-POS*) or triplets (*child node-POS, dependency relation, parent node-POS*). They encode features using a histogram vector containing the number of POS, NE or syntactic relation pair types.

[20] `https://spacy.io/`

To acquire contextual information, they experiment with multiple types of word embeddings such as GloVe [34] embeddings trained on Twitter and Wikipedia, word2vec embeddings trained on Google News, FastText [30] embeddings trained on various sources and transformer BERT embeddings. According to observation in [12], different layers of BERT capture different kinds of information. Therefore, to extract one embedding per tweet, they experiment with various pooling strategies to find the appropriate one. The experiments encapsulate four distinct combinations: a concatenation of the last four hidden layers, averaging the last four hidden layers, the last hidden layer, and the second to last hidden layer. The final embedding is normalised so that the L2 norm of the vector is 1. To finalise the overall representation of the tweet, they concatenate all the syntactic features with either BERT or other types of embedding and then apply PCA for dimensionality reduction.

Cheema et al. perform a grid search over PCA energy conservation, regularization parameter C and gamma of RBF kernel during training of the SVM model. Their best scoring model with 0.7217 MAP is an ensemble of SVM models that utilises POS, syntactic dependency relations and BERT embeddings.

2.4.7 Martinez et al.

Similarly to Cheema et al. [7], Martinez et al. [29] also participated in 2020 CheckThat! Lab [2]. However, they submitted approaches for tasks 1, 2 and 5. Both tasks 1 and 5 deal with the check-worthiness problem, although on different topics, Covid-19 and politics, and consequently different datasets.

The dataset for task 1 is the same as described in subsection 2.4.6. The dataset for task 5 is comprised of 50 fact-checked documents such as debates, speeches, press conferences, and other similar formats in the training subset and 20 debates in the test subset.

Martinez et al. extract two types of features: embedding and graph. The embeddings are either self-generated during training or preloaded at startup. The latter uses Twitter GloVe embeddings of dimension 200[6]. They tokenize the tweets using the NLTK tokenizer to use the first 50 tokens as the input for the embeddings. As for the graph features, they utilise the complete information about each tweet provided by the authors of the dataset to increase the size of the input with tweets related to the given tweet. They use the complete information to extract triples in format (*tweet-relation-subject*) such as (*tweet-quoted-tweet*) or (*tweet-reply_status-tweet*) and construct a graph from these triples. For each tweet, they iterate over its edges in the graph to find a relation node which is in turn connected to at least three other tweet nodes. After such nodes are found, they concatenate their tweet text to the text of the original tweet. They prepare a version of the English Regressive Imagery Dictionary (RID) with a format compatible with liwc[21] Python module to perform text analysis. The dictionary is comprised of 3,150 words and roots in 48 categories, which are further divided into three main categories: pri-

[21] `https://pypi.org/project/liwc/`

mary, secondary and emotion. The input vector based on this dictionary contains one number per category representing the percentage of words in those categories.

For both tasks 1 and 5, Martinez et al. propose five different models with slight variations based on the task. The first four models take as input word embeddings of the first n words of the tweet. Model 1 is a feed-forward neural network (FFNN). The architecture after the input layer continues with a 1D global max-pooling layer that operates on n word vectors, a hidden layer, and a sigmoid layer of size 1, which outputs a score between 0 and 1. Model 2 is of type CNN with the architecture that consists of the following layers: input, embedding, two pairs of convolutional 1D with a kernel size of 5 and max-pooling 1D with a pool size of 5, flatten, dense, and finally, a dense sigmoid layer of size 1. Model 3 is a long short-term memory network (LSTM) with input, embedding, LSTM and dense sigmoid layers. The fourth model is a bi-directional LSTM model with a comparable architecture as Model 3, but instead of one LSTM layer and one dense layer, it has two bi-directional LSTM layers and two dense layers. Finally, Model 5, like Model 1, is FFNN, although with input constructed from tf-idf vectors. The network contains three pairs of dense layers that scale down by 50% in each stage and batch normalization layers, followed by a batch normalization layer, a dropout layer and a dense sigmoid layer. Models 2, 3 and 4 are unaffected by the task they are applied to. However, Models 1 and 5, applied to task 5, run without a hidden layer but with input enriched with the vectors described at the end of the previous paragraph.

Martinez et al. conducted a grid search to find the best performing hyperparameters configuration. For task 1, the best performing configuration is Model 4 (Bi-LSTM) with both graph and embedding features, tangent activation function, 10 neurons in the hidden layer, no dropout, 64 epochs, batch size of 10 and the MAP score of 0.7425. As for task 5, the best performing configuration is Model 4 (Bi-LSTM) with no oversampling, embedding features, tangent activation function, 256-neuron hidden layer, 0.2 dropout, 10 epochs, 16 batches, and finally, the MAP score of 0.17. The official results from 2020 CheckThat! Lab show that their primary and constrastive1 runs based on the Bi-LSTM model with GloVe embeddings achieved the first positions in the classification.

2.5 Analysis summary

Shapiro et al. [38] define journalism as a “discipline of verification." Although fact-checking and verification are often used interchangeably, Silverman [39] sees fact-checking as a “specific application of verification in the world of journalism." We could say that verification is the necessary step in the process of fact-checking. A new but related meaning for fact-checking emerged in 2003 with the launch of FactCheck.org, a website which goal is to “monitor the factual accuracy of what is said by major U.S. political players in the form of TV ads, debates, speeches, interviews and news releases." Since then, a number of fact-checking organisations has been on the rise, reaching over 300 in 2020. The process of manual fact-checking is non-trivial and requires a substantial amount of time. In the case of PolitiFact, this amount of time can be estimated, ranging from one to three days per fact-check.

According to Vosoughi et al. [45], who investigated the spread of true and false information on Twitter, "falsehood diffused significantly farther, faster, deeper and more broadly than the truth in all categories of information." Furthermore, Leskovec [27] found a typical lag between peaks of attention to a phrase in the news media and blogs is 2.5 hours. Combining these two pieces of information with the aforementioned estimate of time needed to complete one fact-check, it becomes evident that manual journalistic fact-checking cannot keep pace with the spread of misinformation.

Hence, the need for an automated fact-checking system arises. A typical automated fact-checking pipeline consists of three crucial steps: choosing check-worthy sentences, obtaining information related to those sentences, and finally inferring the veracity of the check-worthy sentences based on related information [17].

Currently, one is able to find many works and approaches to the broader task of automated fact-checking. These, however, may vary by their definition of the task being automated. One such task, often described as the first step in an automated fact-checking pipeline, is check-worthiness. As defined by the CLEF 2018 CheckThat! Lab [32], check-worthiness is the prediction task of prioritising claims for fact-checking.

Yearly CheckThat! Lab by CLEF [32, 13, 2], starting in 2018, motivated numerous teams to submit their work on this task. Consequently, an abundance of such works from recent years is at one's disposal for analysis. However, to better grasp the evolution of solutions to check-worthiness task, there are works unrelated to CheckThat! Lab present in the analysis as well.

The collection of analysed works uses a sentence as an input type, with the exceptions being Hansen et al. [17], who used a double representation of a word and works that expect the entire tweet as their input. Similarly, the outputs of analysed approaches vary only slightly. Most approaches output a value in the continuous range between 0.0 and 1.0, while Patwari et al. [33] and Konstantinovskiy et al. [24] only mention values check-worthy or not and claim or not claim, respectively.

The datasets used in analysed approaches can be split into two categories: the authors' and the provided datasets. Teams Hassan et al. [20, 21], Patwari et al. [33], Konstantinovskiy et al [24]. and Hansen et al. [17] created their own datasets, while Zuo et al. [46], Cheema et al. [7] and Martinez et al. [29] used datasets provided by the CheckThat! Lab organizers. Although Hansen et al. [17] competed in CheckThat! Lab too, their work analysed here was published independently and describes their original dataset. The datasets in the latter category differ with the year and topic but are otherwise uniform in their format. Datasets from CheckThat! Lab 2018, 2019 and 2020 [32, 13, 2] (for task 5) consist of political debates transcripts split into sentences, formatted into tab-separated values and annotated with 0 or 1 based on their check-worthiness. CheckThat! 2020 Lab dataset for task 1, tweets on the topic of Covid-19, is comprised of tweets, their metadata, and annotation regarding their check-worthiness. Dataset is provided in `.tsv` and `.json` formats. Datasets created by the authors are similar in having political debates as their source but differ in the methods by which they were created and annotated. Hassan et al. [21] extracted their dataset from US presidential debates and annotated it using a self-developed ground-truth collection website where they of-

fered monetary rewards to annotators to incentivise high-quality labelling. Patwari et al. [33] used US presidential and primary debates and Donald Trump's Presidential Announcement Speech, which they annotated with a self-devised mechanism based on information from multiple fact-checking organizations. Dataset by Konstantinovskiy et al. [24] was created from the transcripts of UK political TV shows and annotated using the crowdsourcing method. Unlike the rest of the approaches, they used an annotation schema with seven different categories. Finally, Hansen et al. [17] created two out of three datasets used in their approach. Both their Embedding Training Dataset and Weakly Labelled Dataset were created using data from American Presidency Project[22]. The latter was weakly annotated using ClaimBuster [21] API[23].

Approaches towards feature extraction deviate across analysed works in the number of different features extracted, i.e. ClaimBuster [21] extracts five, TATHYA [33] eight, and Zuo et al. [46] over 15, and in features selected. Nevertheless, features like part-of-speech (POS), sentiment, and named entities appear in almost all works, although their effectiveness fluctuates, improving TATHYA's SVM model by 4% but unaffecting logistic regression model by Konstantinovskyi et al. [24]. Noteworthy are features based on syntactic dependency parsing utilised by both Zuo et al. [46] and Hansen et al. [18, 17], who won during their respective CheckThat! Lab participation. The most influential features, however, are those based on word embeddings. According to a comparison performed by Hansen et al. [17], the use of word embeddings improved model performance by 5% in the mean average precision (MAP) score. Furthermore, another comparison by Hansen et al. shows that two models which did not utilise word embeddings scored the lowest. The effectiveness of word embeddings could be explained by their ability to capture the linguistic distributional hypothesis that words with similar meanings tend to appear in similar contexts [19].

As previously mentioned, check-worthiness is a classification task, and therefore each approach employs one or more classifiers for deciding on a statement being check-worthy. The most frequently occurring is the Support Vector Machine (SVM) type classifier, as it is used in all but two approaches. Zuo et al.[46] utilised SVM for both classification and feature selection. The latter is noteworthy since they turned the usually disadvantageous vanishing gradient effect into a valuable discriminator of weakly correlated features. There is no readily observable correlation between the specific type of classifier and positive or negative results. This phenomenon could be explained by looking at how different teams used the same type of classifier differently. For example, both Hassan et al.[21] and Patwari et al.[33] used SVM, but Patwari et al.[33] trained multiple SVMs on grouped data, while Hassan et al. trained single SVM without data grouping. Furthermore, as previously outlined, extracted features also influence check-worthiness prediction ability, further obscuring the impact of classifier selection.

[22] `https://web.archive.org/web/20170606011755/http://www.presidency.ucsb.edu/`

[23] `https://idir.uta.edu/claimbuster/api/`

2.6 Implications for our work

During analysis, we have identified potential problems embedded within the check-worthiness task, however, analysed works served us as a potent source of inspiration for solutions to those problems. The proposed approach is comprised of these inspirations as well as original ideas in the hope of achieving improvement in the check-worthiness task.

The proposed approach extracts POS tags and syntactic dependencies as features. We considered extracting more features like named entities or sentiment features, but according to Konstantinovskiy et al. [24] who extracted both POS tags and named entities, the latter shows little promise in improving models scores. The models which utilise POS tags score better in multiple works [20, 33, 46, 24]. Syntactic dependencies appear in works of Zuo et al. [46] and Hansen et al. [17] whom both have won during their respective participation in CheckThat! Lab. According to Hansen et al. [17], syntactic dependencies are able to identify the role of a word in modifying the semantics of other words in the sentence. This is contrasted with word embeddings which encode the semantics of the word in its context.

As previously mentioned in the analysis summary, word embeddings seem to be the most influential feature in improving models' scores. Their choice is, therefore, a significantly important decision. In recent years BERT-based models started showing their superior performance and hence are a good candidate for an embedding extraction tool. Sentence embeddings are also worth considering, as shown by Konstantinovskiy et al. [24] who utilised InferSent[24] sentence embeddings to great success.

To battle a heavy class imbalance and inspired by Hansen et al. [17], we will use so-called weak labelling on a large additional dataset. Dataset will be acquired from American Presidency Project via means of web scraping. We will weakly label it using ClaimBuster online check-worthiness api[25].

Finally, the type of classification model will depend on the level of embeddings used. While for word embeddings, we will train a bi-directional LSTM model to capture the interactions between word embeddings in a sentence. For sentence embeddings, a model consisting of a short sequence of dense layers will probably suffice as the majority of inter-word interactions are captured within sentence embeddings.

With such a proposed approach, we hope to outperform at least a few of the newer analysed solutions.

[24] `https://github.com/facebookresearch/InferSent`
[25] `https://idir.uta.edu/claimbuster/api/`

Chapter 3

Datasets used

This section talks about data used in our experiments, source, format, class balance, and minimal pre-processing required to work with the data.

As the main training dataset, we use one that is created, labelled and provided by an entity other than us, in this case, the CLEF association. This decision seems natural as the creation and labelling of our own dataset, we feel, is out of the scope of this thesis. Furthermore, the CLEF association's dataset satisfies all the expected requirements: domain specificity, thorough labelling, real-world imitation, and considerable size.

3.1 Political debates dataset

The CLEF team provided this dataset for the participants of 2019 CheckThat! Lab task 1 [1]. The dataset available on GitHub[1] is comprised of American political speeches and debates from the years 2015 through 2019.

The data is split into `test`, `test_annotated` and `training` directories, which are in turn split into multiple tab-separated values (`.tsv`) files. Each `.tsv` file corresponds to and holds the transcripts of the specific political speech or debate. There are 15,554 and 6,478 sentences in all of the files inside the training and test directories. The files are named using the date of the event followed by the speaker's name, place of the event or other distinctive features. The files in directories `test_annotated` and training contain four columns: *ID of the sentence*, *source or speaker of the sentence*, *the sentence itself* and *label of the sentence*.

In order to work with the data more efficiently, we wrote a function to combine all of the files in the respective directories into two files representing test and training data. We handled the IDs of the sentences, which would cease to be unique after combining as they are monotonically increasing numbers starting from one in each of the events files by prepending the date of the event to them.

To create a validation dataset, we further split the training dataset with the ratio of three to one in favour of the training dataset. We also create the development (dev) set

[1] `https://github.com/apepa/clef2019-factchecking-task1`

Figure 3.1: The class imbalance in all of the splits of the Political Debates dataset. The percentage of check-worthy sentences in the dataset ranges between 2.1% and 2.9%, indicating a severe class imbalance. The dataset splits from the left top corner to the right bottom corner: training data, training data minus the validation split, test data, and validation split sampled from training data. The pie charts sizes correspond to the sizes of the splits. The fractions of check-worthy and check-unworthy sentences are displayed in red and blue colours, respectively.

by sampling 10% of the training dataset. We use the dev dataset only during the development phase as it is just a fraction of the training dataset, enabling faster workflow.

The data in all the datasets is highly imbalanced, with a fraction of worthy-labelled sentences floating between 2 and 3%. Figure 3.1 displays the class imbalance for all of the datasets in percentage as well as raw values.

3.2 Weakly labelled dataset

Inspired by Hansen et al.'s [17] successful utilization of a weakly labelled dataset, we prepared a dataset containing US presidential candidates' debates from 1960 to 2020. We used the web-scraping technique to obtain the transcripts from the American Presidency Project webpages[2].

We pre-processed the web-scraped content to separate the sources from their sentences and to prepare the sentences for labelling. Some of the separated sources were malformed as a result of inconsistencies present in the source notation. We used the

[2] `https://www.presidency.ucsb.edu/`

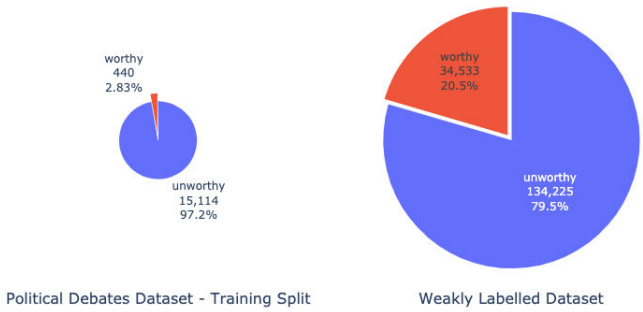


Figure 3.2: The class imbalance and size comparison of Political Debates and Weakly Labelled datasets. The class imbalance in the Weakly Labelled dataset is approximately seven times less severe than in the Political Debates dataset. The Weakly Labelled dataset is also almost eleven times bigger. The datasets from left to right: Political Debates dataset training data before validation split, Weakly Labelled dataset. The pie charts sizes correspond to the sizes of the splits. The fractions of check-worthy and check-unworthy sentences are displayed in red and blue colours, respectively.

Claimbuster web API[3] to weakly label the dataset. The API accepts a single sentence or a paragraph of text, which it splits into sentences and gives each sentence a continuous score between 0 and 1, thus prompting us to introduce a threshold, based on which we decide whether the sentence is check-worthy (1) or not (0). Finally, we export the sentences, their IDs, sources and labels to the `.tsv` files with similar nomenclature described in the section above.

Prior to training with this dataset, we combined all of the files and handled the IDs of the sentences in the same manner applied to the political debates dataset. The dataset contains a total of 168,758 sentences, with 34,533 labelled as check-worthy, which makes up to 20%. Even though the class imbalance is present, it is approximately seven times less severe compared to the political debates dataset. The comparison of the weakly labelled dataset and the political debates dataset's training split is shown in the following figure.

[3] `https://idir.uta.edu/claimbuster/api/`

Chapter 4

Method for fact-check worthiness estimation

4.1 Our Solution

Our solution consists of two independent pipelines, each based on a different approach to check-worthiness task. The first model is centred around the bidirectional LSTM layer, which uses tensors of BERT-based word embeddings as its sentence representation. The second model relies on a BERT-based sentence transformer as its embedding model, followed by a simple neural network.

We also extract text features such as sentence length, less and more granular POS tags and syntactic dependencies to help our sentence representations convey meaning. The features are encoded either using one-hot or sum encodings.

4.1.1 Features

In our solution, we extract multiple features to improve the performance of our models. Prior to feature extraction, we perform the necessary pre-processing steps such as tokenization, POS-tagging and syntactic dependency parsing. We also experiment with the removal of stopwords.

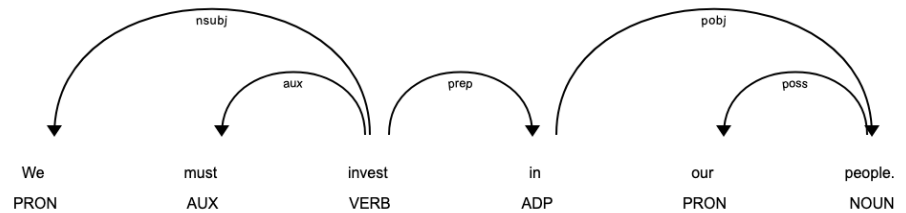


Figure 4.1: Syntactic dependency graph of the sentence "We must invest in our people." The arrows represent dependencies between words they connect, while the origins of the arrows correspond to the head-words and the destinations correspond to the child-words. The role of head-words is to modify the semantics of their child-words.

*Table 4.1: POS tags granularity comparison. The words **you** and **our** share the same less granular POS tag "PRON" but have different corresponding more granular POS tags, "PRP" and "PRP$". The same applies to the words **said** and **undermine**. The meanings of specific POS tags are displayed in rows labelled as "explanation" under corresponding POS tags.*

granularity		**words**			
		you	***our***	***said***	***undermine***
less	**tag**	PRON		VERB	
	explanation	pronoun		verb	
more	**tag**	PRP	PRP$	VBN	VB
	explanation	pronoun, personal	pronoun, possesive	verb, past participle	verb, base form

Two of our feature types are based on part-of-speech (POS) tags of different granularity. The first of the two is extracted by assigning each word in the sentence a tag from the set of Universal POS tags listed on the Universal Dependencies website[1]. The second one is extracted in the same manner, although it assigns a tag from a more granular set of tags. For example, consider the sentence "Secretary Clinton, you have said that it would undermine who we are as Americans, shutting our doors.", particularly the pairs of pronouns "you, our" and verbs "said, undermine". We show the differences in granularity of the two tags sets on the selected words in table 4.1. We encode these feature types using one-hot and sum encodings, giving us four features. For both encodings, we create a zero vector of length equal to the size of the given tag set. We then replace zeroes with ones for one-hot encoding and sums for sum encoding at the indexes corresponding to the tags present in the sentence.

We also utilize tags acquired by syntactic dependency parsing the sentences to create two feature types reflecting the effect of words in the sentence on the semantics of other words in the same sentence. The complete set of tags is listed and described on the Universal Dependencies website[2]. The simpler feature type of the two is created by substituting the words in the sentence with their corresponding dependency tags. The more sophisticated feature type, which we call the triplet feature type, consists of the sequence (*POS tag of the head of a word - dependency tag of a word - POS tag of a word*). The head of the word A is the word B, which modifies the semantics of its child word A. The relationship between the head-word and its child-word is referred to as syntactic dependency. To illustrate these relationships, we provide a syntactic dependency graph of the sentence "We must invest in our people." in figure 4.1. The relationships, or dependencies, are illustrated as arrows pointing from the head-words to their child-words. The triplet feature is the only word-level feature that we extract, which means it is extracted from and appended to each word's embedding. Both of the feature types described in this paragraph are one-hot encoded.

[1] `https://universaldependencies.org/u/pos/`

[2] `https://universaldependencies.org/u/dep/`

We curate a tag selection for all of the previously mentioned tag types by analyzing each tag's discriminatory ability, defined as a combination of the tag's minimal check-worthy and check-unworthy fractions difference and minimal percentage of occurrences in the dataset. Minimal values for both requirements were found empirically using a balanced development subset. By experimenting with this, we hope to create shorter but more discriminatory features. The figure 4.2 shows the values of less granular POS tags for both requirements.

4.1.2 Embeddings

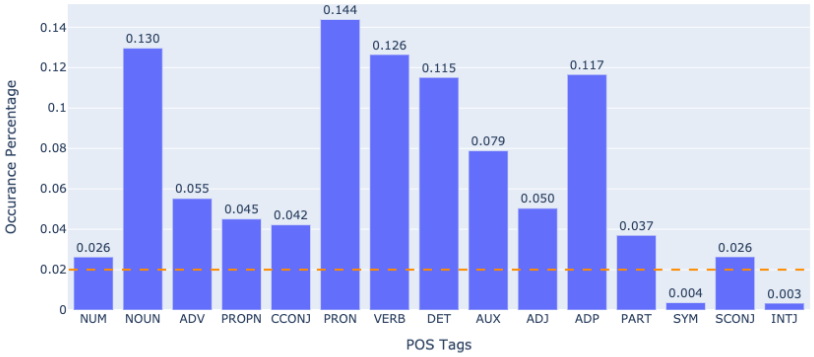
Our solution explores two different embeddings approaches, both based on Bi-directional Encoder Representations from Transformers (BERT) language model created and published by Devlin et al. [12] at Google. The original English version of BERT published in 2018 consisted of two models with multiple encoders and attentions heads, pre-trained on a total of 3,300 million records of unlabelled data. It has since been extended with numerous modified models for different tasks, purposes and languages.

The first approach utilises a pre-trained BERT uncased base model to extract word-level embeddings. The model's intended uses are masked language modelling or next sentence prediction in its raw state and token classification or question-answering in its fine-tuned state. However, the model was pre-trained on a vast amount of text data and consequently acquired an ability to differentiate the words within sentences by their semantics. Since the model was not explicitly pre-trained as a word-embedding model, using its original output as embedding is not recommended. Instead, when loading and initializing the `BertModel` object, we use the `output_hidden_states` option to acquire access to all of the hidden states of the model. We then experiment with three different pooling strategies to extract the embeddings. The pooling strategies are: *extract second to the last hidden layer*, *sum the last four hidden layers*, and *concatenate the last four hidden layers*. After one of the pooling strategies is applied to the model's hidden states, the embeddings are created.

In the second approach, we extract sentence-level embeddings using models from SentenceTransformers framework[3] for state-of-the-art sentence, text and image embeddings. The framework is based on the Sentence-BERT work by Nils Reimers and Iryna Gurevych [37], in which they present a modification to the original BERT model that uses siamese and triplet network structures to extract semantically meaningful sentence embeddings with reduced time consumption. While extracting embeddings, we treat the model as a black box, meaning we input the sentences and expect the embeddings as an output with no additional work. However, we do experiment with three different pre-trained models: `all-MiniLM-L6-v2`, `multi-qa-mpnet-base-dot-v1`, `all-mpnet-base-v2`. The models differ in size, encoding speed, and performance, as noted in the framework's web page table[4].

[3] `https://www.sbert.net/`

[4] `https://www.sbert.net/docs/pretrained_models.html`



(*a*) *POS tags occurrence percentages. The dashed orange line represents a minimal percentage of occurrences in the dataset, which is one of the requirements for a POS tag to be considered discriminatory. The occurrence percentage for a tag* p*,* ν_p*, is calculated as follows:* $\nu_p = \frac{n_p}{n}$*, where* n_p *is the number of all occurrences of tag* p *and* n *is the number of all the tags extracted.*

(*b*) *POS tags check-worthy and check-unworthy fractions differences. The dashed orange line represents a minimal check-worthy and check-unworthy fractions difference, which is one of the requirements for a POS tag to be considered discriminatory. The difference for a tag* p*,* Δ_p*, is calculated as* $\Delta_p = abs(\frac{n_{pw}}{n_p} - \frac{n_{pu}}{n_p})$*, where* n_p *is the number of all the occurrences of tag* p*,* n_{pw} *is the number of tags* p *extracted from check-worthy sentences and* n_{pu} *is the number of tags* p *extracted from check-unworthy sentences.*

Figure 4.2: Figures (*a*) *and* (*b*) *display the two metrics we use to find the most discriminatory tags along with minimal values and values of all the less granular POS tags for these metrics. The minimal values were found empirically. Application of the minimal values for these metrics produces the following subset of less granular POS tags: NUM, NOUN, PROPN, ADJ.*

4.1.3 Models

In our solution, we train two separate classification models, one for each type of embeddings described above.

Bi-LSTM Model

Our first embedding model transforms sentences into sequences of word-level embeddings and thus defines the classification task as sequence classification. We base our model on a Long Short Term Memory (LSTM) layer, a type of Recurrent Neural Network (RNN) with a more sophisticated cell structure, to leverage its capability to learn long-term dependencies present in the sentences.

The bi-directional LSTM layer is followed by an attention mechanism layer inspired by the Kaggle submission[5], and the discussions on the PyTorch forum[6]. The modified attention mechanism we apply is calculated as follows:

$$
\begin{aligned}
Watt_t &= i_t @ Watt_{t-1} \\
ra &= max(0, Watt_t) \\
a &= \frac{exp(ra_{ii})}{\sum_{ij} exp(ra_{ij})} \\
o &= i_t \frac{a_{ii}}{\sum_{ij} a_{ij}}
\end{aligned}
$$

where $Watt_t$ are the attention weights at time t, $Watt_{t-1}$ are the attention weights at time $t-1$ or the initial attention weights at time 0, i_t is the input to the attention layer at time t, @ is the matrix product, ra is the attention tensor after rectified linear unit (ReLu) function application, a is the attention tensor after softmax function application and finally o is the output of the attention layer. The attention weights at time 0 are initialized using the Xavier uniform distribution described in [15]. We regularize the attention layer's output with the dropout layer before feeding it to the optional sequential and dense linear layers. A visually descriptive diagram of the model and its corresponding embeddings model can be seen in the figure 4.3.

Sent-NN Model

Since the sentence-level embeddings model abstracts away the complexities of classifying the sequences of word-level embeddings, our second classifier can remain simple in its structure. It consists of an optional sequential layer and a linear, fully connected layer. The input to both layers is regularized by the dropout layers. The sole purpose of this simple neural network is to classify the output of our sentence-level embeddings model as check-worthy or not. A diagram depicting the model with its corresponding embeddings model is shown in figure 4.4.

[5] `https://www.kaggle.com/code/dannykliu/lstm-with-attention-clr-in-pytorch/notebook`

[6] `https://discuss.pytorch.org/t/self-attention-on-words-and-masking/5671`

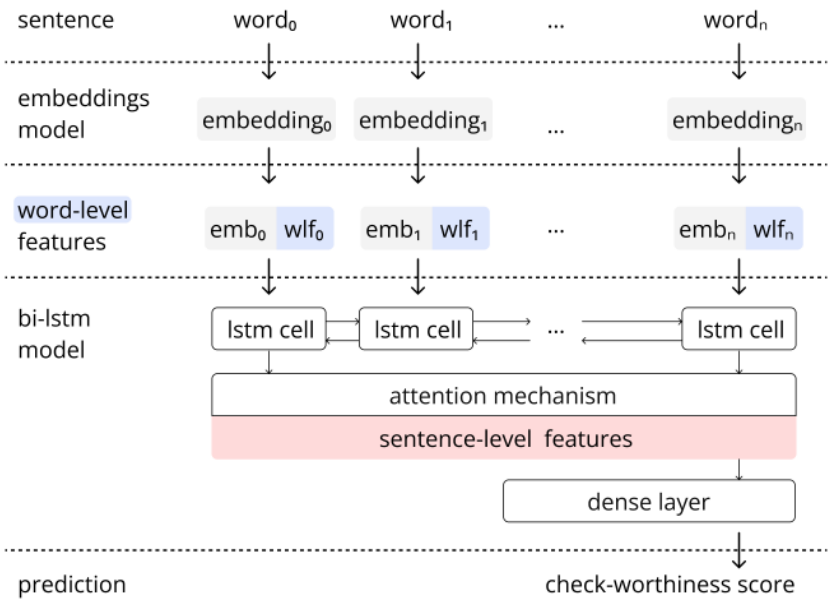


Figure 4.3: Bert Embeddings Model - Bi-LSTM Model diagram. Bert Embeddings Model tokenizes each sentence on its input and encodes each word as an embedding. Optionally, word-level features, highlighted in blue, are appended to word embeddings before the embeddings reach the Bi-LSTM Model. Within the Bi-LSTM Model, the LSTM layer is applied to the word embeddings in both directions, indicated by bi-directional arrows between LSTM cells. The LSTM layer's output is then fed to the attention mechanism before the sentence-level features are optionally appended. Finally, this representation is regularized with the dropout layer and fed to the fully connected layer. The regularization and optional sequential layers are omitted from the diagram for simplicity.

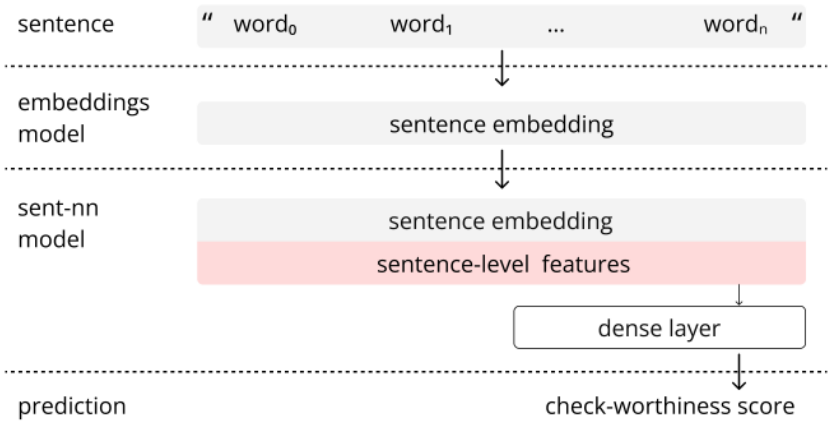


Figure 4.4: Sentence Transformers Embeddings Model - Sent-NN Model diagram. The quotes in the front and back of the sentence indicate that the embeddings model takes an entire unedited sentence as its input. The embeddings model produces a single sentence embedding per sentence. In the Sent-NN model, sentence-level features are appended to the sentence embedding before the tensor is regularized and fed to the fully connected layer. The red background on the sentence-level features indicates their optionality. The regularization layers and optional dense layers are omitted from the diagram for simplicity.

4.2 Implementation

This section introduces essential or otherwise note-worthy implementation aspects and challenges of this work.

4.2.1 Weakly labelled dataset

As previously mentioned in the section 3.2, we acquired data for weak labelling using the web-scraping technique and then labelled it with the Claimbuster API. We web-scraped the data using the Python library Beautiful Soup[7], which allowed us to target specific HTML elements holding the data on the webpage. The challenge here was to handle as many edge cases as possible while separating the sources from their transcripts, as source notation was not uniform across the debates. Even though we have not managed to handle all of the edge cases, we ensured to favour the text's integrity over the source's integrity, meaning we rather let the text overflow into the source than vice-versa. Favouring the text's integrity is crucial as source overflowing into the text could affect the text's labelling. Furthermore, we do not use the source for training or any other purposes and keep it solely for compatibility reasons; hence malformed instances of the source pose no issues.

4.2.2 Baseline methods

This sub-section describes the process of customizing two separate instances of publicly available code on the topic of check-worthiness to enable it to run on our machines and serve as the baseline for our experiments. The code authors, Cheema et al. [7] and Martinez et al. [29], created their code as a part of a submission to the check-worthiness task of the CheckThat! Lab 2020. Both codebases were customized to work with Covid-19 related tweets and political debates datasets, even though the approach by Cheema et al. initially only worked with the former.

Cheema et al.

The source code repository by Cheema et al., available on GitHub[8], consists of two directories: `task_1` and `task_2`. Since the `task_2` directory contains the implementation of their solution to the second task of CheckThat! Lab 2020 [2] on the topic of claim retrieval, it is not discussed any further.

Within the `task_1` directory, there are `format_checker` and `scorer` directories that contain code for format validation and evaluation, respectively, and eleven Python files with various purposes. The `format_checker` and `scorer` code was provided by the CLEF team. Eleven Python files can be divided into two categories, utility and experiments. The utility category consists of three files containing pre-processing, embeddings extraction and helper functions code, respectively. The code in these files is

[7] `https://beautiful-soup-4.readthedocs.io/en/latest/`

[8] `https://github.com/cleopatra-itn/claim_detection`

separated by its purpose and re-used in multiple places, contrasted by the experiments category, which contains vast amounts of duplicate code. Even though different experiments are divided into eight files, named according to format `model-type_embeedings-type_features-type.py` (e.g. `svm_bert_posdep.py`), the differences between the contents of the files are marginal. Prior to the realization of the lack of separation of concerns within experiments files, the amount and complexity of the code appeared challenging to understand. However, once we discovered that the majority of the code shares the same purpose, we were able to form a general understanding and proceed to edit the code to enable it to run on our machine. It should be noted that since this code is part of the submission to CheckThat! Lab competition, separation of concerns was not the top priority for the authors.

Our first goal in customizing Cheema's code was to run it on our CPU-only machine, first with the Covid-19 related tweets dataset and then the political debates dataset. As mentioned before, their approach was intended to only work with the Covid-19 tweets dataset and thus, the changes required to achieve that were few and relatively simple. They consisted of replacing the thundersvm SVM model with a sci-kit SVM model and correcting multiple file paths. We replaced the thundersvm to avoid a complicated set-up paired with no added benefit as thundersvm is only useful on GPU available machines.

Conversely, customization required to run their solution on the political debates dataset was significant. In general, we decided to extract the raw code from scripts into methods with arguments so that they would be callable from other scripts, thus enabling easier experimentation. We edited tweets pre-processing script to act as a method, which accepts string argument denoting the type of dataset altering the behaviour of the method in multiple places (e.g. loading corpus for word segmentation and spell correction for the text pre-processor). Furthermore, instead of using all of the experiment scripts to run different experiments, we extracted and generalized the code within the `svm_bert.py` script so that multiple experiments could be run and controlled from a single point. The size of the political debates dataset combined with limitations of CPU-only machine's performance prompted us to write a dataset sampling method, with which we could sample a development subset with a specified fraction of the size of the original dataset. We also had to edit the provided scorer code to make it work with the political debates dataset, although the required changes were minor, consisting of conditional reading of output files based on the type of dataset.

Once all the necessary customizations had been done, we moved the project to our machine with GPU available for model acceleration. Even though GPU accelerated SVM models are not as common as their neural network counterparts, it can be done using tools like thundersvm[9]. At this point, we planned to utilise the previously replaced thundersvm model, but unfortunately, we encountered an issue stemming from version incompatibility between thundersvm library and CUDA interface on our machine. We explored many different angles to solve this issue, only to be repeatedly stopped by the lack of control over the machine's environment and, consequently, the CUDA version.

[9] `https://github.com/Xtra-Computing/thundersvm`

Martinez et al.

Similarly to Cheema et al. [7], the source code by Martinez et al. is also available at GitHub[10]. It consists of four directories, although we only studied and customized two of them, namely `T1En` for task 1 English version and `T5En` for task 5 English version. The other two directories contain code for task 1 Arabic version and task 2 English version and will not be discussed further.

Martinez et al. chose a different approach to structuring their codebase compared to Cheema et al. [7]. Both directories are comprised of the main script and resources and utils directories. The code within the utils directories is further separated into multiple script files based on its function. The code is well organized and easy to understand. Both task directories also contain `format_checker` and `scorer` used for evaluation and provided by the CLEF team.

The customizations required to run Martinez et al.'s solution were few and trivial in nature. They consisted of installing missing dependencies such as liwc library, fixing IO paths and modifying function for listing train and test data directory to omit files created by us. The numerous models trained in this solution are selected at the end of the main `task5.py` file.

4.2.3 Feature extraction

We have decided to extract features freshly every time we train instead of once before the first training. This decision enabled us to experiment with extraction more freely at the expense of more computing time. During the training, we iterate over batches of sentences, their ids, labels and features provided by the instance of `DataLoader` class from the PyTorch library[11]. The `DataLoader` object expects an instance of the PyTorch `Dataset` class or classes that extend it. We extended the `Dataset` class with our `DebatesDataset` class to modify the returned value to our needs. The `Dataset` class and classes that extend it provide optional argument transform, which expects either an object of the transform class that extends PyTorch `Module` or TorchVision `Compose` object. These transform classes then modify the data before it is pulled from the `Dataset` object by the `DataLoader` object during the training process. We implement feature encoding methods as forward class methods of our custom transform classes and chain these together in various arrangements via TorchVision `Compose` object. We provide the following transform classes: `HandleStopwords`, `OneHot`, `Sum`, `CountWords`, `NoTransform`, `ToBinary` and `ToTensor`. The `HandleStopwords` transform either removes stopwords or not and always occupies the first slot in the `Compose` object. The `ToBinary` and `ToTensor` transform are always used in the last two slots in this particular order to ensure the format of the features. The `OneHot`, `Sum` and `CountWords` are feature transforms. Finally, the `NoTransform` transform simulates an absence of the feature and exists for feature selection purposes.

[10] `https://github.com/jrmtnez/NLP-IR-UNED-at-CheckThat-2020`
[11] `https://pytorch.org/`

4.2.4 Bi-LSTM model

We implemented the Bi-LSTM model and the rest of the models used in this work with the PyTorch library. The Bi-LSTM model's implementation was particularly tricky as it presented multiple challenges. Firstly, the model expects a batch of embeddings padded to the same length, thus requiring us to modify the embedding model to pad the embeddings and return an additional tensor with their original lengths. The model requires lengths as they are used in the PyTorch `pack_padded_sequence` method, significantly reducing computing time by packing the padded input, effectively avoiding non-essential operations on the tensor's padded part. Lastly, since we used a bi-directional version of the LSTM layer, the LSTM layer was applied to the input twice, once in both ways. The resulting output doubled in size, and the part corresponding to the backward application was reversed. Furthermore, we unpacked the LSTM layer's output with the PyTorch `pad_packed_sequence` method to make it compatible with our attention mechanism applied subsequently. Advanced indexing was required to access the correct parts of the output while ignoring the padding zeros.

4.2.5 Hyper-parameter optimization and feature selection

We optimize hyper-parameters and select features using the Optuna library[12], whose main purpose is optimization supported by a wide range of samplers based on various optimization algorithms. The library utilizes three main parts to control the optimization: a study object, a trial object and an objective function. The study object represents one entire optimization experiment. It expects an objective function as an argument, to which it provides trial objects as the optimization proceeds. The trial object represents one iteration with a unique set of values from a specified hyper-parameter or feature space in the optimisation process. The trial objects are created and managed by the study object. Lastly, the objective function contains the actual training code consisting of data loading, models initialization, training and validation iterations and returns a value of a metric being optimized. The trial object is accessible within the objective function and provides a so-called "suggest" interface with methods like `suggest_float` and `suggest_categorical`, with which we define a hyper-parameter and feature space for each optimized hyper-parameter and feature type. We optimize hyper-parameters mostly with the Tree-structured Parzen Estimator (TPE) sampler and select features with the grid sampler[13]. Optuna provides pruners to abort unpromising trials prematurely based on the metric returned by an objective function. However, we chose to implement the `EarlyStopping` class, which does so based on validation loss to avoid values from over-trained models.

[12] `https://optuna.org/`

[13] `https://optuna.readthedocs.io/en/stable/reference/samplers.html`

Chapter 5

Experiments: benchmarking and method evaluation

The chapter describes the purpose and methodology of conducted experiments, results and comparisons, and the discussion regarding our intuitions about the results.

5.1 Experimental methodology

We conduct numerous experiments to assess the effectiveness of various methods and approaches regarding feature selection and hyper-parameter optimization and to test the performance of different versions of our solution on its own as well as compared to other solutions. We conduct the feature selection and hyperparameter optimization in an identical environment, based on the Optuna optimization tool standards, and with a similar methodology, although with different search spaces. We sub-sample the dataset with the same degree of class imbalance and the size of 20% of the original dataset. Even though both feature selection and hyperparameter optimization are conducted in the same environment and, therefore, could potentially be conducted as a single joined experiment, we have decided to separate the two into multiple smaller experiments to avoid unnecessarily large search space and long runtime.

We optimize hyperparameters of both models without and with features. For all of the optimization processes, we use the TPE sampler [1], which, unlike the Grid sampler, chooses values for hyperparameters based on previous values' success, to maximize model's f1-score on positive class. We have reconsidered our initial optimization metric, recall on positive class, due to the negative impact on positive class's precision. It also chooses values for some hyperparameters from continuous ranges and thus requires a number of trials, after which the optimization process is considered completed. We have lowered this number from the initial 200 to 100 to reduce the runtime while still enabling the sampler to find the best hyperparameters.

[1] `https://bit.ly/3GOMKFO`

Our feature selection approach consists of running multiple optimizations over feature parameters space to find the best parameters for each of the less (POS) and more (TAG) granular POS tags and syntactic dependencies (DEP) feature types. The feature parameters search space contains the following parameters: feature type tag set (full or selection described in 4.1.1, feature encoding transform (`OneHot`, `Sum` or `NoTransform` explained in 4.2.3), stopwords (remove them or not), `CountWords` transform (use it as an additional feature or not). Once we have found the best parameters for each of these feature types, we perform optimization processes to find the best combination of features for each model.

After optimizations have been completed, we train the models with the best hyperparameters on the full dataset and both with and without the features. We optimize both models' binary cross-entropy loss function using the Adam optimizer. We train for 30 epochs; however, we use an early-stopping method with validation loss to avoid overfitting. Our early-stopping implementation saves a model checkpoint each time validation loss improves. It also monitors the model's positive class f1-score on the validation data and highlights a checkpoint if the f1-score improves. We implement this behaviour as, during some experiments, the `sent-nn` model's validation loss keeps steadily below the training loss due to the use of dropout regularization layers, which are not activated during validation. At the same time, its f1-score on the positive class grows steadily to a certain point, after which it plummets. This behaviour most likely means that the model learns to ignore positive class as doing so will not increase its loss when training on a highly unbalanced dataset. Once trained, we compare our models' performances with Cheema et al.'s[7] and Martinez et al.'s[29] baseline methods described theoretically in 2.4.6, 2.4.7 and practically in 4.2.2.

In the last set of experiments, we train the models on a combination of original training data and ample amounts of weakly labelled training data. We combine the original data with the weakly labelled data in three different manners, resulting in different-sized datasets and different positive class ratios within them. We then compare models trained on these datasets with our best models trained on the original dataset to confirm or reject our surmise that big amounts of lower quality data can improve the model's performance.

5.2 Results

The following section presents the results our models were able to achieve with and without extracted features. We also show a comparison with selected baseline methods followed by results achieved on weakly labelled data. We score models' performances using three metrics: average precision, f1-score on positive class and accuracy. These metrics were chosen to provide a fairground for comparisons as the baseline methods optimized average precision, our models optimized f1-score on positive class and accuracy reflects the general ability of the models to classify.

Table 5.1: The effect of text feature extraction on the models' performances. The comparison includes our model based on the bidirectional LSTM layer (`bi-lstm`), which utilizes word-level embeddings and our simple neural network using sentence-level embeddings (`sent-nn`). The models are evaluated after training without features and with sentence-level features. Additionally, the performance after training with word-level features is added for `bi-lstm` model. The comparison considers average precision, f1-score on positive class, and accuracy as its metrics. The models' versions marked with "" are loaded from the highlighted checkpoints, explained in the section 5.1, as opposed to the checkpoint with the lowest validation loss. Our best performing model is the featureless `sent-nn` model.*

metrics	**models**				
	bi-lstm			***sent-nn***	
	no features	*sent-level features**	*word-level features**	*no features**	*sent-level features**
avg. precision	0.0504	0.0470	0.0638	**0.1522**	0.1233
pos. class f1	0.0751	0.0513	0.0664	**0.1477**	0.1289
accuracy	0.5503	0.3419	0.6521	**0.8250**	0.8119

5.2.1 The effect of features

We test the features' effect on our models' performances by training the models with the same hyperparameters with and without the use of features. The results of this testing are summarised in Table 5.1. The number of experiments is uneven because our model based on a bidirectional LSTM layer (`bi-lstm`) supports word- and sentence-level features, while our sentence embeddings neural network only works with sentence-level features.

Our best performing model is the `sent-nn` model without the use of text features, which outperforms itself with text features and all three versions of the `bi-lstm` model in all three metrics with 15.22% average precision, 14.77% f1-score on positive class and 82.50% accuracy. The second best, `sent-nn` with features, loses 2.89%, 1.88% and 1.31%, respectively. Focusing only on the `bi-lstm` model, its word-level version scores the best in average precision and accuracy, while the best f1-score is achieved by the version without text features. However, both versions are underperforming, with more than 6% losses on the best model in average precision and f1-score and approximately 20% losses in accuracy.

5.2.2 The comparison with baseline methods

For this comparison, we select both our models' best performing versions. In the case of the `bi-lstm` model, it is the version utilizing word-level features, while for the `sent-nn` model, the featureless version is selected. The results are displayed in the Table 5.2.

Our featureless `sent-nn` model outperforms all other models in average precision and f1-score, scoring 15.22% and 14.77%, respectively and is only superseded in accuracy, with 82.5%, by Martinez et al.'s and Cheema et al.'s methods. The second-best scoring method in average precision is one by Martinez et al., losing 2.62% on our `sent-nn`

Table 5.2: The comparison with baseline methods. We compare the best-performing versions of our models with methods by Cheema et al. and Martinez et al. in average precision, f1-score on positive class and accuracy. These metrics should provide a fairground for comparison as both baseline methods optimized average precision, our models optimized the f1-score, and accuracy reflects a general ability of the models to classify. Our `sent-nn` *model outperforms all other models in average precision and f1-score. Martinez et al.'s model scores the best in accuracy.*

metrics	**models**			
	Cheema et al.	***Martinez et al.***	***our bi-lstm***	***our sent-nn***
avg. precision	0.1086	0.1260	0.0638	**0.1522**
pos. class f1	0.1053	0.0100	0.0664	**0.1477**
accuracy	0.9764	**0.9800**	0.6521	0.8250

model. It also performs the best in accuracy with 98.0%, beating the second-best Cheema et al.'s method by 0.36%. Our `bi-lstm` model lags behind all other models, outperforming only Martinez et al.'s method in f1-score on the positive class by 5.64%.

5.2.3 The effect of weakly labelled data

In this experiment, we test the effect of training on a combination of weakly labelled data and original training data as a potential solution to issues regarding class imbalance and a general scarcity of the data. The best performing featureless `sent-nn` model is trained on four datasets created using distinct merging strategies. The strategies, their corresponding product dataset size and positive class ratio, along with the results our best model achieved, are displayed in Table 5.3.

The table shows that combining training data with weakly labelled data does not improve the model's performance, as the best dataset to train on is the original one with no weakly labelled data combined. Out of the merged datasets, the model performs best on the one created by the simple merging strategy, with an average precision of 12.96%, losing 2.26% on the model trained on the original data. In terms of the f1-score on positive class, the merged datasets yield similar results in the range between 10.08% and 10.45%.

5.3 Discussion and results interpretation

In this section, we present our opinions and intuitions on the results introduced in the previous section 5.2, their potential causes and possibilities for improvement. We also discuss additional experiments done after all the steps from the methodological procedure described in section 5.1 have been completed.

Firstly, we address the subpar performance of our `bi-lstm` model. We suspect the model could benefit from multiple LSTM layers instead of only one that it has now. Simultaneously, the fully connected layers following the LSTM layer could be simplified, supported by the experiment where we inactivated an additional sequential layer

Table 5.3: The effect of merging weakly labelled data into the original dataset on our featureless `sent-nn` model's performance. The first column shows metrics achieved on the original dataset. The rest of the columns represent three datasets obtained using distinct merging strategies. The `balanced original` strategy uses only positive weakly labelled samples to equal the number of negative samples in the original dataset, hence the positive class ratio of 50%. When applying the `simple` merging strategy, we merge the entirety of both datasets into one. The resulting dataset is the largest and borrows the positive class ratio of approximately 20% from the weakly labelled dataset. The `balanced result` strategy is realized by calculating the sum of positively labelled samples from both datasets and matching it with an equal number of negative samples. As mentioned in the subsection 5.2.3, using weakly labelled data in this manner did not improve our `sent-nn` model's performance.

metrics	**different merging strategies**			
	none	***balanced original***	***simple***	***balanced result***
# of sentences	11665	22654	140473	54285
pos. class ratio	0.0280	0.5000	0.1926	0.5000
avg. precision	**0.1522**	0.0762	0.1296	0.1184
pos. class f1	**0.1477**	0.1011	0.1008	0.1045
accuracy	**0.8250**	0.7387	0.6993	0.7058

and the attention layer. After these modifications, the model's average precision has improved by 8.26% in the featureless version and 10.02% in the sentence-level features version. It should be noted that while average precision has improved, the f1-score on positive class retained its previous values or even dropped. We think this points to the model's focus shifting on a highly prevalent negative class.

Secondly, we attribute the success of our `sent-nn` model to its use of sentence embeddings. The idea of utilising sentence embeddings comes from Konstantinovskiy et al. [24], who used InferSent, sentence embeddings created by passing GloVe word embeddings through the BiLSTM network, and a simple logistic regression model to outperform their baseline methods. Furthermore, the transformers-based models have recently been successfully applied in various NLP tasks, signalling their superior ability to convey meaning.

Both of our models' versions that utilize sentence-level text features score worse than their featureless counterparts. This could be attributed to the structure of our pipeline, which extracts embeddings and features in batches directly during training. Such structure enables us to experiment with and make necessary changes to the extraction process and lifts the requirement to store the embeddings and features on the disk but restricts access to certain metadata, such as the range of values within the embeddings tensors. This restriction further restricts us to scale embeddings meaningfully, which in turn lessens the potential effect of features as their vectors contain values from the narrower range between 0 and 1.

Next, we would like to discuss the performances of our models compared to baseline methods. As mentioned in the subsection 5.2.2, our `sent-nn` model is the best performing in average precision and f1-score on positive class, but in the accuracy, it is only the

third-best behind both baseline methods. The best performing model in accuracy is the one by Martinez et al., which at the same time scores the lowest in f1-score on positive class. This indicates that the model focuses primarily on the negative class, similarly to our `bi-lstm` model after the additional simplifying experiment described in the second paragraph of this section. Interestingly, both models are in their core bidirectional LSTM networks.

The last item to address is the effect of merging weakly labelled data into the original dataset on the model's performance. Across all three merging strategies, we see that they yield no improvement. Our intuition is that merging weakly labelled data with humanly labelled data confuses the model due to potentially high amounts of faulty labels present in the weakly labelled dataset. A more effective way of using weakly labelled data seems to be the pre-training approach, where one pre-trains the model on the weakly labelled data and then fine-tunes it using the original training data. We conducted an extra experiment with this approach by pre-training featureless versions of `bi-lstm` and `sent-nn` models on various fractions of the weakly labelled dataset and fine-tuning the pre-trained model on the original dataset.

Chapter 6

Conclusion

In this thesis, we tackled the problem of identifying the statements that potentially contain societally relevant and therefore check-worthy claims. The problem is generally known as the *check-worthiness task*. We thoroughly analysed the existing solutions, taking inspiration from them in the process, to learn about the effectiveness of the various previously applied techniques and models. Two of the analysed works had their corresponding code publicly available, which allowed us to use them as our baseline methods.

Many of the analysed works were created to be used as a submission to CheckThat! Lab competition by CLEF association. This competition, which challenges academic teams in various NLP tasks, also provides domain-specific humanly labelled datasets for its participants. For training and evaluation, we used the 2019 CheckThat! Lab's heavily imbalanced dataset consisting of 15,554 and 6,478 politically related sentences in training and test splits.

Inspired by the analysed works of Hansen et al. [17] and Konstantinovskiy et al. [24], we designed and implemented two different check-worthiness models, `bi-lstm` and `sent-nn`. The `bi-lstm` solution is a neural network based on a bidirectional LSTM layer, which utilizes BERT-based word embeddings as its text representation. The `sent-nn` solution takes a different path by utilizing transformers-based sentence embeddings as its text representation, which conveys meaning very well and therefore strips the classifier of otherwise necessary complexities.

These models were subjected to extensive experimentation. We tested their base performance and the effect of extracted text features on it. We found that our best performing model is the featureless `sent-nn` model. The model also outperformed both selected baseline methods. It gained more than 2.5% in average precision over Martinez et al.'s solution and more than 4.2% in f1-score on positive class over Cheema et al.'s. The best version of our `bi-lstm` was the one utilizing word-level text features. However, the model generally underperformed. The inclusion of the sentence-level features did not yield any performance improvement. We also tested the effect of merging the original training data with weakly labelled data, unfortunately with no performance gains.

Although both of our models could be improved upon, we consider the `sent-nn` model a contribution to the field as it considerably outperformed baseline methods using state of the art technologies. Future work would consist of fixing models' obvious shortcomings, experimenting with different strategies of incorporating weakly labelled data and reimplementing the models into end-to-end solutions.

Bibliography

[1] Pepa Atanasova et al. "Overview of the CLEF-2019 CheckThat! Lab on Automatic Identification and Verification of Claims. Task 1: Check-Worthiness". In: *Experimental IR Meets Multilinguality, Multimodality, and Interaction*. Ed. by Fabio Crestani et al. Cham: Springer International Publishing, 2019, pp. 301–321. ISBN: 978-3-030-28577-7.

[2] Alberto Barrón-Cedeño et al. "Overview of CheckThat! 2020: Automatic Identification and Verification of Claims in Social Media". In: *Experimental IR Meets Multilinguality, Multimodality, and Interaction*. Ed. by Avi Arampatzis et al. Cham: Springer International Publishing, 2020, pp. 215–236. ISBN: 978-3-030-58219-7.

[3] Hannah Bast, Björn Buchhold, and Elmar Haussmann. *Overview of the Triple Scoring Task at the WSDM Cup 2017*. 2017. arXiv: 1712.08081 [cs.IR].

[4] Hannah Bast, Björn Buchhold, and Elmar Haussmann. "Relevance Scores for Triples from Type-Like Relations". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '15. Santiago, Chile: Association for Computing Machinery, 2015, pp. 243–252. ISBN: 9781450336215. DOI: 10.1145/2766462.2767734. URL: https://doi.org/10.1145/2766462.2767734.

[5] Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 747–754. DOI: 10.18653/v1/S17-2126. URL: https://aclanthology.org/S17-2126.

[6] David M Blei, Andrew Y Ng, and Micheal I Jordan. "Latent Dirichlet Allocation". In: *Journal of Machine Learning Research* 3 (2003), pp. 993–1022.

[7] Gullal S. Cheema, Sherzod Hakimov, and Ralph Ewerth. *Check_square at CheckThat! 2020: Claim Detection in Social Media via Fusion of Transformer and Syntactic Features*. 2020. arXiv: 2007.10534 [cs.CL].

[8] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. arXiv: 1409.1259 [cs.CL].

[9] Giovanni Luca Ciampaglia et al. "Computational Fact Checking from Knowledge Networks". In: *PLOS ONE* 10.6 (June 2015), pp. 1–13. DOI: 10.1371/journal.pone.0128193. URL: https://doi.org/10.1371/journal.pone.0128193.

[10] Alexis Conneau et al. "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: Sept. 2017, pp. 670–680.

[11] Leon Derczynski et al. *SemEval-2017 Task 8: RumourEval: Determining rumour veracity and support for rumours*. 2017. arXiv: 1704.05972 [cs.CL].

[12] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[13] Tamer Elsayed et al. "Overview of the CLEF-2019 CheckThat! Lab: Automatic Identification and Verification of Claims". In: *Experimental IR Meets Multilinguality, Multimodality, and Interaction*. Ed. by Fabio Crestani et al. Cham: Springer International Publishing, 2019, pp. 301–321. ISBN: 978-3-030-28577-7.

[14] William Ferreira and Andreas Vlachos. "Emergent: a novel data-set for stance classification". In: June 2016, pp. 1163–1168. URL: http://www.politifact.com/.

[15] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[16] Lucas Graves. "Anatomy of a Fact Check: Objective Practice and the Contested Epistemology of Fact Checking". In: (2016). DOI: 10.1111/cccr.12163.

[17] Casper Hansen et al. "Neural Check-Worthiness Ranking with Weak Supervision: Finding Sentences for Fact-Checking". In: *Companion Proceedings of The 2019 World Wide Web Conference*. WWW '19. San Francisco, USA: Association for Computing Machinery, 2019, pp. 994–1000. ISBN: 9781450366755. DOI: 10.1145/3308560.3316736. URL: https://doi.org/10.1145/3308560.3316736.

[18] Casper Hansen et al. "Neural weakly supervised fact check-worthiness detection with contrastive sampling-based ranking loss". In: vol. 2380. 2019. URL: http://ceur-ws.org/Vol-2380/paper_56.pdf.

[19] Zellig S. Harris. "Distributional Structure". In: *WORD* 10.2-3 (1954), pp. 146–162. DOI: 10.1080/00437956.1954.11659520. eprint: https://doi.org/10.1080/00437956.1954.11659520. URL: https://doi.org/10.1080/00437956.1954.11659520.

[20] Naeemul Hassan, Chengkai Li, and Mark Tremayne. "Detecting Check-Worthy Factual Claims in Presidential Debates". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. Melbourne, Australia: Association for Computing Machinery, 2015, pp. 1835–1838. ISBN: 9781450337946. DOI: 10.1145/2806416.2806652. URL: https://doi.org/10.1145/2806416.2806652.

[21] Naeemul Hassan et al. "Toward Automated Fact-Checking: Detecting Check-Worthy Factual Claims by ClaimBuster". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1803–1812. ISBN: 9781450348874. DOI: 10.1145/3097983.3098131. URL: https://doi.org/10.1145/3097983.3098131.

[22] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[23] Garth S Jowett and Victoria O'donnell. "What is propaganda, and how does it differ from persuasion?" In: *Propaganda and Misinformation*. Sage publications, 2006.

[24] Lev Konstantinovskiy et al. *Towards Automated Factchecking: Developing an Annotation Schema and Benchmark for Consistent Automated Claim Detection*. 2018. arXiv: 1809.08193 [cs.CL].

[25] Bill Kovach and Tom Rosenstiel. *The Elements of Journalism, Revised and Updated 4th Edition: What Newspeople Should Know and the Public Should Expect*. Crown, 2021.

[26] Dieu-Thu Le, Ngoc Thang Vu, and Andre Blessing. "Towards a text analysis system for political debates". In: Association for Computational Linguistics, Aug. 2016, pp. 134–139.

[27] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. "Meme-tracking and the Dynamics of the News Cycle". In: (2009).

[28] Alexios Mantzarlis. *Will verification kill fact-checking?* Oct. 2015. URL: https://www.poynter.org/fact-checking/2015/will-verification-kill-fact-checking/.

[29] Juan Martinez-Rico, Lourdes Araujo, and Juan Martinez-Romo. "NLP&IR@UNED at CheckThat! 2020: A Preliminary Approach for Check-Worthiness and Claim Retrieval Tasks using Neural Networks and Graphs". In: vol. 2696. 2020.

[30] Tomas Mikolov et al. *Advances in Pre-Training Distributed Word Representations*. 2017. arXiv: 1712.09405 [cs.CL].

[31] Ndapandula Nakashole and Tom M Mitchell. "Language-Aware Truth Assessment of Fact Candidates". In: June 2014, pp. 1009–1019. URL: http://www.w3.org/TR/rdf-primer/.

[32] Preslav Nakov et al. “Overview of the CLEF-2018 CheckThat! Lab on Automatic Identification and Verification of Political Claims”. In: *Experimental IR Meets Multilinguality, Multimodality, and Interaction*. Ed. by Patrice Bellot et al. Cham: Springer International Publishing, 2018, pp. 372–387. ISBN: 978-3-319-98932-7.

[33] Ayush Patwari, Dan Goldwasser, and Saurabh Bagchi. “TATHYA: A Multi-Classifier System for Detecting Check-Worthy Statements in Political Debates”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM ’17. Singapore, Singapore: Association for Computing Machinery, 2017, pp. 2259–2262. ISBN: 9781450349185. DOI: 10.1145/3132847.3133150. URL: https://doi.org/10.1145/3132847.3133150.

[34] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “GloVe: Global Vectors for Word Representation”. In: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. URL: https://aclanthology.org/D14-1162.pdf.

[35] John Platt et al. “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods”. In: *Advances in large margin classifiers* 10.3 (1999), pp. 61–74.

[36] Hannah Rashkin et al. “Truth of Varying Shades: Analyzing Language in Fake News and Political Fact-Checking”. In: Sept. 2017, pp. 2931–2937.

[37] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: Nov. 2019. URL: https://arxiv.org/abs/1908.10084.

[38] Ivor Shapiro et al. “Verification as a Strategic Ritual”. In: *Journalism Practice* 7 (2013). ISSN: 1751-2794. DOI: 10.1080/17512786.2013.765638. URL: http://dx.doi.org/10.1080/17512786.2013.765638.

[39] Craig Silverman. “Verification and Fact Checking”. In: *Verification Handbook: A definitive guide to verifying digital content for emergency coverage*. Ed. by Craig Silverman. 1st ed. European Journalism Centre (EJC), 2014. URL: https://datajournalism.com/read/handbook/verification-1/additional-materials/verification-and-fact-checking.

[40] Mark Stencel and Joel Luther. *Fact-checking count tops 300 for the first time*. Oct. 2020. URL: https://reporterslab.org/fact-checking-count-tops-300-for-the-first-time/.

[41] James Thorne and Andreas Vlachos. “An Extensible Framework for Verification of Numerical Claims”. In: Apr. 2017, pp. 37–40. URL: http://herox.com/factcheck.

[42] James Thorne and Andreas Vlachos. “Automated Fact Checking: Task formulations, methods and future directions”. In: (June 2018). URL: http://arxiv.org/abs/1806.07687.

[43] James Thorne et al. *FEVER: a large-scale dataset for Fact Extraction and VERification*. 2018. arXiv: 1803.05355 [cs.CL].

[44] Andreas Vlachos and Sebastian Riedel. "Identification and Verification of Simple Claims about Statistical Properties". In: Sept. 2015, pp. 2596–2601. URL: https://github.com/.

[45] Soroush Vosoughi, Deb Roy, and Sinan Aral. *The spread of true and false news online*. 2018. DOI: 10.1126/science.aap9559. URL: http://science.sciencemag.org/.

[46] Chaoyuan Zuo, Ayla Karakas, and Ritwik Banerjee. "A Hybrid Recognition System for Check-worthy Claims Using Heuristics and Supervised Learning". In: vol. 2125. 2018. URL: https://par.nsf.gov/biblio/10162052.

Appendix A

Technical documentation

The following chapter briefly introduces the reader to the technical details of this thesis and provides the necessary information to run the code if the occasion arises.

Our original code, as well as baseline methods code, is available on GitHub and accessible by clicking the following links. Please note that for Cheema et al.'s and Martinez et al.'s solutions, we provide two links for each; the first one is their original code, and the second one is the code that we forked and modified to run with the political debates data and on our machines.

- our code: `https://github.com/icobx/dt`
- Cheema et al.:
  - original: `https://github.com/cleopatra-itn/claim_detection`
  - modified: `https://github.com/icobx/claim_detection`
- Martinez et al.:
  - original: `https://github.com/jrmtnez/NLP-IR-UNED-at-CheckThat-2020`
  - modified: `https://github.com/icobx/NLP-IR-UNED-at-CheckThat-2020`

The political debates dataset used to train and evaluate our solution is also available on GitHub.

- `https://github.com/icobx/clef2019-factchecking-task1`

The code is written entirely in Python, either in the form of Jupyter notebooks or as standard `.py` files. The required versions for associated tools are listed below.

- `python>=3.7.4`
- `jupyter_core>=4.7.1.`
- `ipython>=7.25.0`

- ipykernel>=5.5.5

The works utilize various Python packages. Those that are essential are listed below and in requirements_v1.txt file in our repository[1].

- beautifulsoup4==4.10.0
- camel-tools==1.2.0
- ekphrasis==0.5.1
- huggingface-hub==0.0.19
- matplotlib==3.4.3
- nltk==3.6.3
- numpy==1.21.2
- optuna==2.10.0
- pandas==1.3.3
- plotly==5.5.0
- requests==2.26.0
- scikit-learn==1.0
- scipy==1.7.1
- sentence-transformers==2.2.0
- spacy==3.1.3
- stanfordnlp==0.2.0
- torch==1.9.1
- torchvision==0.10.1
- tqdm==4.62.3
- transformers==4.11.3

[1] https://github.com/icobx/dt/blob/master/requirements_v1.txt

A.1 Cheema et al.

Since Cheema et al.'s solution was originally implemented for a dataset consisting of Covid-19 related tweets, running the original code requires a considerable amount of modifications to it. Therefore, we recommend running our modified version. The code can be found in `task_1` directory. Please note that the solution is not optimized well and can run for a long time. To run the solution, please follow these instructions:

1. change the paths to the downloaded dataset in `definitions.py`
2. run `combine_datasets` and `preprocess` methods found in `preprocess_data.py` file
3. run `extract_bert_embeddings` method in the file of the same name
4. finally, run `svm_bert` method in `svm_bert.py` file.

A.2 Martinez et al.

Within Martinez et al.'s repository, the directory `T5En/nlpir01` contains code relevant to the check-worthiness task. The solution required very few modifications to run in our setting; however, it is more convenient to run our modified version of the solution. To run the solution, please follow these instructions:

1. change the paths to the downloaded dataset in `utils/global_parameters.py`
2. within `task5.py` file, either uncomment the lines corresponding to CLI setup at the bottom of the file and control the model, embeddings and other parameters via command line or modify the parameters directly in the file.
3. run the `task5.py`.

Please note that when running the solution for the first time, the user may be asked to download additional dependencies and should proceed according to the instructions provided.

A.3 Our solution

Most of our solution's code is written in Jupyter notebooks and has an experimental character and therefore is not very user-friendly. The optimalization code is located in `model_bi_lstm_optim.ipynb` and `model_sent_nn_optim.ipynb` notebooks, although we do not recommend running it as it requires a lot of internal knowledge to run properly. The training portion of the code is located in `model_bi_lstm_training.ipynb` and `model_sent_nn_training.ipynb` notebooks. To run the solution, please follow these instructions:

1. change the paths to the downloaded dataset in `definitions.py`
2. run the `run` method in `model_bi_lstm_training.ipynb` or `model_sent_nn_training.ipynb` notebooks with only one required argument `is_training`, set to `True`
3. run the same method again with `is_training` set to `False`

The `run` method provides various arguments, although an introduction to their functionality is outside of the scope of this documentation. Please note that when running the solution for the first time, the user may be asked to download additional dependencies and should proceed according to the instructions provided.

Appendix B

Súhrn diplomovej práce v slovenskom jazyku

B.1 Úvod do problematiky

Nepravdivé informácie sa prostredníctvom sociálnych médií šíria rýchlejšie, a zároveň k viacerým používateľom než informácie pravdivé, ako je ukázané tímom Vosoughi et al.. V kombinácii s faktom, že viac ako 60 % ľudí získava informácie a správy na sociálnych médiách, to vytvára legitímne nebezpečenstvo pre ľudstvo ako celok. Organizácie ako FactCheck.org, PolitiFact alebo Demagóg sa pokúšajú zmierniť potenciálne škody spôsobené nepravdivými informáciami manuálnym a komplexným fact-checkingom (fact-checking - overovanie pravdivosti výrokov na základe faktov, ďalej len“overovanie" “overovanie“) spoločensky relevantných tvrdení nachádzajúcich sa vo verejnom priestore. Avšak, obsiahly a na zdroje náročný proces manuálneho overovania zaostáva za jednoduchým a neviazaným vznikom a rozptylom nepravdivých informácií.

Z toho dôvodu vzniká potreba automatizovaného overovacieho systému. Štandardná sekvencia automatizovaného overovania väčšinou pozostáva z troch hlavných krokov: detekcia častí textu hodných overovania (tzn. vety obsahujúce tvrdenia, ktoré by mali byť overené), zozbieranie informácií relevantných k tvrdeniam hodným overovania a nakoniec použitie týchto relevantných informácií na určenie pravdivosti daných tvrdení. Väčšina výskumu v tejto oblasti sa primárne venovala posledným dvom krokom sekvencie. Tieto kroky, teda zozbieranie relevantných informácií a určenie pravdivosti tvrdení, zväčša očakávajú tvrdenia hodné overovania ako ich vstup. Avšak krok prvý, detekcia častí textu hodných overovania, bol donedávna podstatne menej skúmaný. To zmenil vznik súťaže CheckThat! Lab, iniciovanej asociáciou CLEF, určenej na hľadanie tých najlepších riešení NLP (natural language processing alebo spracovanie prirodzeného jazyka) úloh, akými sú získavanie tvrdení z textu, detekcia falošných informácií a odhad vhodnosti overovania.

Skoršie riešenia sa spoliehajú na pomerne jednoduché tf-idf črty [21, 33], pri reprezentácii ich slov a následne viet, kým novšie práce reprezentujú vety používajúc

podstatne sofistikovanejšie a zmysluplnejšie slovné [46, 17, 7, 29] alebo dokonca aj vetné [24], tzv. embeddingy. Práce rozširujú ich vetné reprezentácie pomocou textových čŕt, ako napríklad dĺžka vety [21, 33, 46], sentiment [21, 33, 46], part-of-speech (POS) označenia [21, 33, 46, 24, 7], menované entity (named entities - NE) [21, 33, 46, 24] a syntaktické závislosti [46, 17], v snahe zakódovať v nich čo najviac významu. Avšak ako sa technológia embeddingov zlepšuje, hodnota textových čŕt klesá, čo má za následok zjednodušovanie vetných reprezentácií neskorších prác v kontexte textových čŕt a čoraz väčšie kladenie dôrazu na embeddingy. Použitie čo najpokrokovejších embedding modelov sa zdá byť najdôležitejším rozhodnutím aké môže jednotlivec urobiť pri návrhu riešenia problému vhodnosti overovania.

Cieľ tejto práce sa skladá z dvoch hlavných bodov:

1. *nájsť, modifikovať, v prípade potreby, a spojazdniť dostupné riešenia problému vhodnosti overenia za účelom získania metód určených na porovnanie,*
2. *navrhnúť a implementovať originálne riešenie schopné dosiahnuť lepšie výsledky ako spomenuté metódy.*

Inšpirovaní prácami z oblasti výskumu vhodnosti overovania a motivovaní faktom, že žiadna predošlá práca nevyužila vetné embeddingy založené na Bidirectional Encoder Representations from Transformers (BERT)[1] jazykovom modeli, trénujeme dva separátne modely využívajúce embeddingy založené na BERT. Prvý model, inšpirovaný prácou od tímu Hansen et al. [17], je založený na využití obojsmernej LSTM vrstvy a slovných embeddingov získaných pomocou základného BERT modelu. Druhý je jednoduchou neurónovou sieťou, ktorá si berie inšpiráciu od tímu Konstantinovskiy et al. a ich použitia vetných embeddingov [24]. S našou slovnou aj vetnou reprezentáciou experimentujeme získavaním čŕt založených na menej a viac granulárnych POS označeniach a syntaktických závislostiach. Trénovanie prebieha na vysoko nerovnomerných dátach (viď obrázok 3.1) zložených z prepisov viacerých politických debát, poskytnutých asociáciou CLEF pre 2019 CheckThat! Lab [1]. V snahe zmierniť problémy spojené s nevyváženosťou tried v datasete a celkovým nedostatkom dát, experimentujeme s trénovaním na podstatne väčšom doménovo špecifickom datasete, ktorý je tzv. slabo anotovaný už existujúcim riešením problematiky vhodnosti overovania.

Za účelom získania metód určených na porovnanie, sme stiahli a upravili dve riešenia problematiky vhodnosti overovania [7, 29], aby dokázali pracovať s dátami, ktoré používame. Výkon našich modelov vyhodnocujeme pomocou tzv. average precision a f1-score na pozitívnej triede, aby bolo vyhodnotenie férové, keďže metódy určené na porovnanie optimalizovali prvú spomenutú metriku a naše modely tú druhú. Náš najvýkonnejší model, ktorý používa vetné embeddingy, ako jeho vetnú reprezentáciu, dosahuje 4.36% and (a) 2.62% zlepšenia, v average precision, nad riešeniami tímov Cheema et al. [7] a Martinez et al. [29] Taktiež dosahuje lepšie výsledky ako obe metódy v f1-score a to o 4.24% and (a) 13.77%.

[1] `https://huggingface.co/docs/transformers/model_doc/bert`

Figure B.1: Porovnanie rôznych množín hlavného datasetu. Percentá overovania hodných viet sa pohybujú v rozmedzí od 2.1 % až 2.9 %, čo značí podstatnú nerovnomernosť anotačných tried. Množiny datasetu z ľavého horného rohu po pravý spodný roh: originálna trénovacia množina, trénovacia množina mínus validačná množina, testovacia množina, validačná množina. Veľkosti koláčových grafov korešpondujú s veľkosťou množín, ktoré reprezentujú. Červenou farbou sú značené overovania hodné časti množín a modrou farbou tie nehodné.

Našim prínosom je klasifikátor typu neurónovej siete, implementovaný pomocou nástroja PyTorch[2], ktorý využíva predtrénovaný vetný transformer na zakódovanie viet do vetných embeddingov.

B.2 Použité dáta

Hlavným datasetom použitým na trénovanie a vyhodnotenie je dataset vytvorený, anotovaný a poskytnutý asociáciou CLEF. Dataset bol poskytnutý ako súčasť súťaže Check-That! Lab 2019 a je zložený z 15,554 trénovacích a 6,478 testovacích viet pochádzajúcich z viacerých politických debát a vystúpení. Debaty a vystúpenia sú reprezentované `.tsv` súbormi, ktoré obsahujú stĺpce: *ID vety, zdroj vety, obsah vety* a *anotáciu vety*. Z trénovacej množiny datasetu sme vyčlenili približne 25 % validačných dát. Anotačné triedy v datasete sú podstatne nevyvážené. Percento overovania hodných viet sa naprieč rôznymi množinami datasetu pohybuje v rozmedzí 2 % až 3 %. Na obrázku B.1 sú znázornené rôzne množiny datasetu, ich veľkosti a percentá overovania hodných viet.

[2] `https://pytorch.org/`

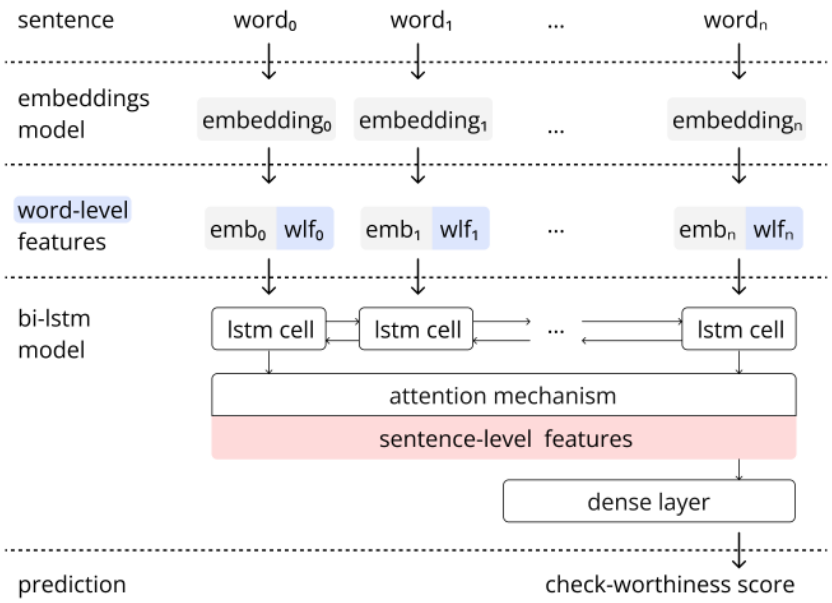


Figure B.2: Diagram sekvencie BERT embeddingový model – Bi-LSTM model. Bert embeddingový model rozdeľuje vety na jeho vstupe na slová a kóduje ich na slovné embeddingy. K tým môžu byť následne prilepené slovné črty. Takto zakódované slová putujú do LSTM vrstvy a z nej následne cez mechanizmus pozornosti do plne prepojenej vrstvy, ktorá je už schopná produkovať výsledok v podobe pravdepodobnosti vhodnosti overenia.

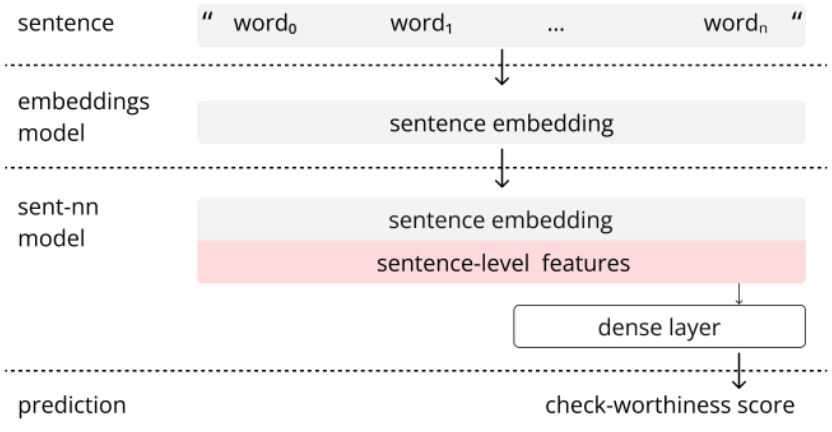


Figure B.3: Diagram sekvencie vetný transformátor – Sent-nn. Úvodzovky na začiatku a konci vety indikujú, že vetný transformátor akceptuje na svojom vstupe celú vetu nerozdelenú na slová. Ten taktiež produkuje jeden vetný embedding pre každú vetu. K takémuto embeddingu sa pripoja vetné črty a následne putuje do plne prepojenej vtstvy, ktorá produkuje výsledok v podobe pravdepodobnosti vhodnosti overenia.

Table B.1: Porovnanie našich modelov: Efekt textových čŕt na výkon modelov. Porovnanie zahŕňa oba naše modely, bi-lstm aj sent-nn. Modely sú vyhodnotené bez použitia textových čŕt, a zároveň aj s ich použitím. V prípade bi-lstm modelu je zahrnutý aj výsledok s použitím slovných textových čŕt.

metriky	**modely**				
	bi-lstm			***sent-nn***	
	bez čŕt	*vetné črty*	*slovné črty*	*bez čŕt*	*vetné črty*
avg. precision	0.0504	0.0470	0.0638	**0.1522**	0.1233
pos. class f1	0.0751	0.0513	0.0664	**0.1477**	0.1289
accuracy	0.5503	0.3419	0.6521	**0.8250**	0.8119

Table B.2

B.3 Naše riešenie

Naše riešenie sa skladá z dvoch samostatných modelov, pričom každý je založený na inom prístupe k problematike vhodnosti overovania. Prvý model je založený na využití obojsmernej LSTM vrstvy, ktorá využíva viacdimenzionálne matice slovných embeddingov založených na jazykovom modeli BERT. Druhý model používa vetný transformátor založený na BERT jazykovom modeli ako jeho embeddingový model. Ten je nasledovaný jednoduchou neurónovou sieťou.

Taktiež extrahujeme textové črty, ako napríklad dĺžka vety, menej a viac granulárne POS označenia a syntaktické závislosti, za účelom zlepšenia schopnosti našich vetných reprezentácií uchovávať význam. Tieto črty sú následne zakódované pomocou one-hot alebo sum kódovaní. Náčrty modelov sú zobrazené na obrázkoch B.2 a B.3.

B.4 Výsledky

V prvých experimentoch porovnávame naše dva modely navzájom, a zároveň vplyv extrahovaných slovných a vetných čŕt na ich výkon. Našim najvýkonnejším modelom je tzv. sent-nn model, teda model využívajúci vetný transformátor a jednoduchú neurónovú sieť bez čŕt. Tento model predbehol všetky ostatné varianty našich modelov vo všetkých troch metrikách, a zároveň predbehol aj obe metódy určené na porovnávanie v metrikách average precision a f1-score na pozitívnej triede. Výsledky sú zhrnuté v tabuľkách B.2 a B.3.

B.5 Záver

V tejto práci sme sa venovali téme identifikácie viet potenciálne obsahujúcich spoločensky relevantné, a teda overovania hodné tvrdenia. Tejto problematike sa zvyčajne hovorí problém vhodnosti overovania (*check-worthiness*). Dôkladne sme analyzovali existujúce

Table B.3: Porovnanie našich modelov s metódami tímov Cheema et al. a Martinez et al. v metrikách average precision, f1-score na pozitívnej triede a accuracy. Tieto metriky by mali poskytovať férové podmienky na porovnanie, keďže metriku average precision optimalizovali riešenia vyššie spomenutých tímov a metriku f1-score na pozitívnej triede optimalizovali naše riešenia. Metrika accuracy v tomto prípade slúži na vyjadrenie všeobecne schopnosti riešení klasifikovať.

metriky	**modely**			
	Cheema et al.	***Martinez et al.***	***náš bi-lstm***	***náš sent-nn***
avg. precision	0.1086	0.1260	0.0638	**0.1522**
pos. class f1	0.1053	0.0100	0.0664	**0.1477**
accuracy	0.9764	**0.9800**	0.6521	0.8250

riešenia, z ktorých sme si v procese brali inšpiráciu. V prípade dvoch analyzovaných prác bol voľne dostupný aj kód ich riešení, ktorý sme využili na získanie metód určených na porovnanie.

Na trénovanie bol použitý voľne dostupný dataset vytvorený asociáciou CLEF pri príležitosti nimi usporiadanej súťaže CheckThat! Lab, ktorej sa zúčastnilo aj viacero tímov s nami analyzovanými prácami.

Inšpirovaní prácami tímov Hansen et al. [17] a Konstantinovskiy et al. [24] sme navrhli a implementovali dve rôzne riešenia problému vhodnosti overovania, `bi-lstm` a `sent-nn`. Riešenie `bi-lstm` je neurónovou sieťou založenou na obojsmernej LSTM vrstve, ktorá využíva viacdimenzionálne matice BERT slovných embeddingov, ako jej vetné reprezentácie. Na druhej strane, riešenie `sent-nn` na získanie vetných reprezentácií využíva vetný transformátor založený na BERT jazykovom modeli a tie následne spracúva jednoduchá plne prepojená vrstva.

Rozsiahle experimenty ukázali, že našim najvýkonnejším modelom je `sent-nn` bez čŕt, ktorý prekonal náš `bi-lstm` model ako aj obe metódy určené na porovnávanie v metrikách average precision a f1-score na pozitívnej triede. Metódu od tímu Martinez et al. [29] prekonal o 2.5 % v metrike average precision a metódu od tímu Cheema et al. [7] o 4.2 % v f1-score na pozitívnej triede. Najlepšou verziou modelu `bi-lstm` je tá, ktorá využíva slovné črty na zlepšenie vetnej reprezentácie. Celkovo však tento model nedosahoval dostatočné výsledky.

Napriek tomu, že pre oba modely existuje priestor na vylepšenia, pokladáme náš `sent-nn` model za úspech, keďže prekonal metódy určené na porovnanie s použitím vetného transformátora, založeného na BERT jazykovom modeli, ako jeho embeddingového modela.