PolitBERT
Deepfake Detection of American Politicians using Natural Language Processing

Master Thesis

Maurice Rupp

University of Bern

March 2021
Abstract

This thesis explores the application of modern Natural Language Processing techniques to the detection of artificially generated videos of popular American politicians. Instead of focusing on detecting anomalies and artifacts in images and sounds, this thesis focuses on detecting irregularities and inconsistencies in the words themselves, opening up a new possibility to detect fake content. A novel, domain-adapted, pre-trained version of the language model BERT combined with several mechanisms to overcome severe dataset imbalances yielded the best quantitative as well as qualitative results. Additionally to the creation of the biggest publicly available dataset of English-speaking politicians consisting of 1.5 M sentences from over 1000 persons, this thesis conducts various experiments with different kinds of text classification and sequence processing algorithms applied to the political domain. Furthermore, multiple ablations to manage severe data imbalance are presented and evaluated.

Supervisor:
Prof. Dr. Paolo Favaro
Computer Vision Group
University of Bern
# Contents

## 1 Introduction
1.1 Motivation ....................................................... 5
1.2 Goals ......................................................... 6
1.3 Contributions .................................................. 6
1.4 Thesis Structure ............................................... 6

## 2 Theoretical Background
2.1 The Perceptron ............................................... 7
2.2 The Concept of Neural Networks .............................. 8
2.3 Attention ....................................................... 10
2.4 Transformer Networks ......................................... 10
2.5 Imbalanced Data ............................................... 12
  2.5.1 Over-/Under-sampling .................................... 12
  2.5.2 Balanced Loss Functions ................................. 12
  2.5.3 Data Augmentation ....................................... 13
     2.5.3.1 Textual Data Augmentation ......................... 13
     2.5.3.2 Latent Space Data Augmentation .................. 13
  2.5.4 Multi-Class to Binary / Ensemble Learning ............ 14
  2.5.5 Common Regularization Techniques .................... 14

## 3 Related Work
3.1 Fake Detection ............................................... 15
  3.1.1 Fake News ............................................... 15
  3.1.2 Deepfakes ................................................. 16
     3.1.2.1 Deepfake Technology ............................... 16
     3.1.2.2 State-of-the-Art in Deepfake Detection .......... 16
     3.1.2.3 Problems with Deepfake Detection ............... 17
  3.2 Text Classification ......................................... 17
     3.2.1 Text Representation .................................... 18
        3.2.1.1 Word Representation .............................. 18
        3.2.1.2 Character Representation ......................... 18
        3.2.1.3 POS Representation ............................... 18
  3.2.2 Authorship Attribution .................................. 19
     3.2.2.1 Traditional Learning Approaches ................ 19
     3.2.2.2 Approaches using Neural Networks ............... 19
  3.3 Language Modeling and Text Generation .................. 20
1 Introduction

1.1 Motivation

Artificially generated media, called deepfakes, have recently gained widespread attention for their application in generating non-consensual pornographic videos of celebrities\[44\], faking statements of politicians\[143\] and other cases of fraud\[114\][43\], revealing the tremendous threats of this rapidly progressing technology. Unfortunately, the generation of deepfakes has already progressed far enough such that laypeople can no longer distinguish real from fake content in certain scenarios\[1\]. A recent example created by the Belgian branch of the Extinction Rebellion showed how simple the spread of such videos actually is. In the deepfaked video, the Belgian Prime Minister Sophie Wilmès talks about a possible link between deforestation and COVID-19\[37\]. Despite the fact that the video comment included a statement of it being fake\[2\], many users believed in the authenticity of the video.

As politicians generally have significant influence on world affairs, they are a vulnerable and impactful target for deepfakes in which they state inauthentic information. Due to the global interconnectivity through social media, false information can be spread efficiently and effectively in a drastically short amount of time.

Although there exist several pathways to detect deepfakes, they can all be countered with adversarial training or other counter-attacks\[92\]. Multiple experts of the field argue that it is only a matter of time until conventional detection algorithms will no longer be able to separate real from fake\[134\]. Therefore, a novel approach to prevent deepfakes from harmfully disfiguring people is in dire need to be developed.

---

1 To get a sense of how hard it is to detect deepfakes, the reader is advised to visit https://detectfakes.media.mit.edu/, an MIT research project about if and how people detect deepfakes.

2 Quote from the Facebook post: “This video may be fake, but the information it contains is genuine.”
1.2 Goals

This thesis’ key idea is to approach deepfake detection from a linguistic opposed to visual and auditory perspectives; i.e. it does not focus on detecting artifacts or other anomalies in the images and sounds of deepfakes, but rather detects irregularities and inconsistencies of the linguistic content. The domain in focus is the American political circle, which, due to Americas global power position and the recent presidential elections, raises immense potential of influencing people by advertising and spreading fake content about presidential candidates and other politicians. Using state-of-the-art NLP\(^3\) tools, the goal is to create an algorithm which can confidently attribute authorship of spoken sentences from a pool of impactful American politicians and correctly classify novel, possibly fake sentences.

Naturally, as it is common in most language problems, there is some amount of ambiguity, which cannot be removed completely, but only reduced. This includes generic phrases such as "Thank you, Ms ..." or "Hello ... it's great to be here!" which will be impossible to attribute to a specific person. Another challenge consists of small semantic changes in a phrase which completely change its meaning, as, for example, adding "not" to a phrase. The third challenge is to find a proper form of discretising text and not losing too much semantic meaning by quantification.

1.3 Contributions

This work’s contributions can be summarized in five points:

- A brief introduction to artificial neural networks focused on NLP as well as the presentation of prior work in deepfake detection, text classification and related fields.

- The biggest public dataset of English-speaking politicians currently available online\(^4\).

- PolitBERT, the first, publicly available, political, domain-specific, pre-trained version of the language model BERT\(^5\).

- Various experiments with existing networks for sequence processing and classification adapted to the task of authorship attribution in the political domain.

- A fine-tuned version of PolitBERT, which outperforms the existing tuned networks at the task of authorship attribution in the political domain.

- Multiple ablation studies using PolitBERT in the context of dealing with heavily imbalanced datasets.

1.4 Thesis Structure

This thesis will be further structured as follows:

In chapter 2, the theoretical framework for this thesis will be established. Besides the general functionalities of neural networks, more specific concepts and techniques, which will be used later on, are introduced. In chapter 3, prior work in the fields of deepfake detection, text classification and language modelling is discussed. Subsequently, chapter 4 presents the data as well as the developed method to the proposed task. Chapter 5 presents experiments with different networks, ablations and quantitative as well as qualitative evaluation of the results. Finally, chapter 6 provides the final conclusion of this thesis, addressing limitations and possible future work.

---

\(^3\) Abbreviated from Natural Language Processing.

\(^4\) https://www.kaggle.com/mauricerupp/englishspeaking-politicians

\(^5\) https://huggingface.co/maurice/PolitBERT
Recent advances in machine learning, especially the area of deep learning have received a massive amount of attention in the broad public. Be it due to algorithms which sort our social media feeds, self-driving cars or advances in several medical domains. Artificial, self-learning intelligence is shaping our world and its society in a fundamental way, continuously revealing the enormous potential and threats of this technology. Neural networks form the basis for most of these approaches. In sections 2.1 and 2.2, I will briefly introduce the general framework and terminology of neural networks. Afterwards, I will elaborate on those concepts which are more closely connected to this thesis in sections 2.3, 2.4 and 2.5.

2.1 The Perceptron

Originating from a neuroscientific approach to simulate the functions of a human brain, researchers started publishing models of brain neurons and their ability to learn in the early Forties. The most groundbreaking work was published in 1943, wherein neurophysiologist Warren McCulloch and mathematician Walter Pitts introduced the McCulloch-Pitts-Neuron, a model of how brain neurons fire. Building on this theoretical framework, Frank Rosenblatt introduced the concept of the perceptron in 1958: a single neuron, which "learns" its weight through several iterations, while minimizing the difference between the desired and actual output. The input is multiplied by the weight and then added to a bias term, which is adjustable as well. In this thesis, the term 'neuron' is used as a synonym to the basic concept of the perceptron.

Later on, researchers proceeded to combine multiple perceptrons in so-called Multilayer-Perceptrons (MLP), which form the origin of neural networks. Usually, each perceptron from a layer is connected to each perceptron of the next layer. This is called a fully-connected layer. The term 'Deep Learning' stems from stringing together multiple layers of neurons and ergo forming a deep neural network, where all the layers which are neither on the input nor output level are called hidden layers, since their output is usually not displayed.
2.2 The Concept of Neural Networks

Independently of the internal architecture of neural networks, there exist several key parts which are briefly summarized in this section. For an in-depth explanation of the topic the reader is advised to consult the Book ‘Deep Learning’ by Goodfellow et al.[48]. Generally speaking, neural networks map numerical, potentially high dimensional inputs to numerical outputs. While inputs are usually referred to as \( X \) for all inputs or \( x \) for a single sample, outputs are referred to as \( Y_{\text{pred}} \) and \( y_{\text{pred}} \). These output values can be interpreted as, e.g., pixel colors, words or probabilities depending on the task at hand. The available data is split into a training set used to adjust the weights of the neurons, a validation set used to determine other parameters and a test set which is used only once at the end to evaluate the performance of the network. A neural network can adjust its weights in a task-oriented manner with the following concepts:

**Activation functions** describe the value propagated by a neuron for a given input value. Usually, such a function is directly applied to each input value individually. Arguably the most popular function for hidden neurons is the ReLU\(^1\) activation function introduced by Hahnloser et al.[104]. It maps negative outputs to zero and positive values to their identity. The activation function used for the last neuron layer of a network strongly depends on which output values one deems appropriate. For example, modeling a binary probability can be done with the Sigmoid activation function, since its outputs lie in \([0,1]\). To model a probability for more than two classes the softmax activation function is usually applied on each value of the output vector:

\[
\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}
\]

All entries of the vector sum up to 1, which can be interpreted as the probability distribution over the \( j \) classes.

\[\text{ReLU}(x)\]

\[\text{Sigmoid}(x)\]

Figure 2.1: Plots of the ReLU and Sigmoid activation functions.

In order for the network to adjust its weights according to an objective, an **error function\(^2\)** measures the performance of the network. It calculates the difference between the desired and the predicted output by the network and returns it as a single numerical value.

---

\(^1\) Abbreviated from *Rectified Linear Unit*.

\(^2\) In the literature, error functions are also referred to as loss or cost functions.
CHAPTER 2. THEORETICAL BACKGROUND

The desired output $Y_{true}$ can either be a given value\(^3\) (supervised learning), a value directly deduced from the data\(^4\) (self-supervised learning) or a transformation of the input in some other measurable way\(^5\) (unsupervised learning). There further exist combinations of these techniques: A variation of transfer learning, for example, is achieved by pre-training\(^6\) a model on an unsupervised task and then fine-tuning in a supervised manner. Transfer learning is generally understood as applying (previously learned) knowledge from one task to a somewhat similar task. The tasks either share some semantics of the input and/or the output. Since all tasks in this thesis can be interpreted as (self-) supervised tasks, I only introduce error functions which apply to this use-case:

- **Binary Cross-entropy:**
  \[
  L_{BCE}(y_{pred}, y_{true}) = -y_{pred} \log(y_{true}) - (1 - y_{pred}) \log(1 - y_{true})
  \]
  The typical loss function to use in a binary problem, e.g. a positive-negative classification. It is usually combined with a sigmoid activation function, in order for the model to output one single numerical value between 0 and 1.

- **(Log/) Cross-entropy loss:**
  \[
  L_{CE}(y_{pred}, y_{true}) = -\sum_{i=1}^{n} y_{pred, i} \log(y_{true, i})
  \]
  A generalized version of binary cross-entropy, which is applied on multi-class predictions with $n$ classes. Its inputs are an $n$-dimensional vector, where each of the $n$ positions can be interpreted as the probability of the sample belonging to the corresponding class and a scalar indicating the correct class. It is usually combined with a softmax activation function to get values in $[0, 1]$ for each entry.

Equipped with a tool to measure the error of a network, an algorithm for processing this information to achieve performance improvement is needed. Nowadays, backpropagation\(^10\) is the most successful algorithm to achieve this goal\([108]\). Recursively using the derivative chain rule, it propagates back the measured error through the network calculating the gradient value for the weight of each neuron.

Adjusting the parameters of the network in order to improve the gradients’ values is achieved by using optimization functions (optimizers). Most state-of-the-art optimizers use a variation of the basic gradient descent algorithm:

\[
\theta = \theta - \alpha \nabla_\theta E
\]

This algorithm takes the error ($E$) of the whole dataset\(^7\) and adjusts the parameter $\theta$, which represents the weights of all neurons, by subtracting the gradient $\nabla_\theta E$. The number $\alpha$ is called learning rate (or step-size) and indicates by which amount the weights of the neurons are shifted to the direction of the optimum. In practice, this procedure is usually not suitable for the whole dataset at once, since the calculation of the gradients would be too complex. Instead, the algorithm is applied to a mini-batch. Mini-batches are disjunct subsets of the dataset, whose conjunction forms the whole dataset. The most commonly used optimization function is the Adam\(^8\) optimizer, which uses a combination of first- and second-order moments of the gradients to adjust the neurons’ weights\([78]\).

I will now introduce several concepts which are more specifically related to the topic of this thesis.

---

\(^3\) E.g. text classification or pixel prediction.
\(^4\) E.g. splitting the input data and trying to predict the missing parts.
\(^5\) E.g. representation learning.
\(^6\) Pre-training is here understood as adjusting the parameters of a network by training on another task than the actual target.
\(^7\) Meaning the error of one iteration through each sample of $X_{train}$.
\(^8\) Abbreviated from Adaptive moments.
2.3 Attention

In deep learning, the concept of attention is generally interpreted as a way of “telling” a network which parts of an input appear the most relevant for the target task. Recently, this mechanism has become an integral part of sequence modelling and transduction models for various tasks, since such models can learn to put emphasis on relevant parts of the input\[7\][76]. Attention itself is usually implemented as a dot-product between two intermediate states/outputs of a network combined with multiple activation functions such as softmax. Afterwards, it is concatenated or multiplied with a “normal” subsequent layer of the network.

Self-attention or intra-attention is a special form of attention, which relates and rearranges different positions of a single sequence in order to gain a more informative representation of the sequence\[19\]. Instead of looking at the whole input at once, self-attention allows the different parts of the input to interact with one another to determine which parts of the input appear relevant and/or connected in this context. Instead of using a single attention function, some works apply multi-head attention, which outputs multiple different attention layers from the same sequence, concatenates those and then applies a normal attention layer again. Multi-head attention therefore allows a model to jointly extract information from different representation subspaces at different positions and can be understood as applying multiple attention filters to an input instead of only one.

2.4 Transformer Networks

Recently, the encoder-decoder structure has proven to be particularly efficient in solving neural sequence transduction problems (e.g. machine translation\[20\]): First, the encoder processes the input into a latent representation\[9\]. Given the latent representation, the decoder then calculates the output. The idea is that the encoder learns to transform the input in a way which simplifies the task for the decoder. This concept is often used in various image processing tasks as well\[36][72\].

A Transformer follows this overall structure with multiple encoder blocks followed by the same amount of decoder blocks. It usually applies self-attention to model dependencies of elements inside sequences with no regard for their distance inside the sequence\[127\]. The original Transformer employs residual connections\[10\] (marked red in figure 2.2), multi-head attention and normal fully-connected (feed-forward) layers.

Since Transformer models contain no recurrence and no convolution, the word order is injected into the network by using positional encodings. This encoding outputs a unique vector for each element in the sequence, depending on its position inside the sequence. The positional encoding is added to a word embedding (I will elaborate on word embeddings in section 3.2.1).

A significant amount of research has been conducted on models using the original Transformer as a basis, since they are able to understand and model long-term dependencies in an input and process vast amounts of data due their capability of parallelization.

---

9 A latent representation of a sample can be understood as a mapping or feature representation in another mathematical space.
10 Residual or skip connections are direct connections between two non-sequential layers of a neural network. The output of one layer is concatenated with the output of a subsequent layer.
Figure 2.2: The original Transformer architecture where $N$ is equal to 6. The output of the last (of the $N$) encoder is fed to all $N$ decoders. Picture taken from [127].
2.5 Imbalanced Data

Having developed the general theoretical framework, I will now have a deeper look at one specific data problem: In real-life scenarios datasets are usually heavily imbalanced, meaning that certain classes occur significantly more frequently than others. This is due to several factors: a class could simply be more present in the environment one collects data from or certain data types could be easier to collect in general. Severe imbalance can lead the algorithms to focus on highly represented classes only and neglect underrepresented ones. The simplest solution seems to be to discard as many samples of overrepresented classes until the quantities are balanced. However, this could lead to biases and weaker generalization for said classes, so it’s usually not considered a valid option.

2.5.1 Over-/Under-sampling

A fairly common approach to handling class imbalance is to randomly over- or under-sample the data. While in cases of over-sampling, instances of underrepresented authors are processed multiple times per epoch, in under-sampling only a random subset of instances of overrepresented authors are considered each epoch. It was argued that even if under-sampling incurs risks from not including important samples, it is still preferred to over-sampling[21].

2.5.2 Balanced Loss Functions

Another approach to tackle the same issue is to use a weighted loss function to deal with the imbalance. The algorithm processes every sample exactly once per epoch as usual, but the loss function assigns different weights to the samples. The most straightforward way in doing so is to assign a higher weight to samples from underrepresented authors, so that they have a greater impact on the loss than those from overrepresented ones. The class-corresponding weight $\alpha_{y_{true}}$ is mostly chosen heuristically or by using the inverse class frequency$^{11}$:

$$L_{CEW}(y_{pred}, y_{true}) = -\alpha_{y_{true}} \sum_{i=1}^{n} y_{pred} \log(y_{true})$$

Focal loss, introduced by Lin et al., combines the weighted cross-entropy with a confidence discount factor, which reduces the relative loss for well-classified examples and puts more focus on hard, misclassified samples[84]. This factor depends on the probability a classifier assigns to the correct class and a hyperparameter $\gamma$, which determines the level of discount. Since a classifier should be less confident with samples from underrepresented authors due to their low occurrence, this helps focusing on such samples. Focal loss is usually combined with class-weighting.

$$L_{focal}(y_{pred}, y_{true}) = -\alpha_{y_{true}} \cdot (1 - y_{pred})^\gamma \sum_{i=1}^{n} y_{pred} \log(y_{true})$$

The final loss function introduced here represents a heuristic of choosing the hyperparameter $\alpha$. So-called class-balanced loss first calculates the effective number of samples for each class and then uses this value as the weighting factor $\alpha$[28]. It can therefore be combined with focal or weighted cross-entropy loss. The hyperparameter $\beta \in [0, 1)$ determines how big the gap between the $\alpha$ of over- and underrepresented classes should be. The closer $\beta$ is to one, the bigger the gap. This way, the effective number of samples for a class $i$ and $k_i$ samples is calculated as follows:

$$E_i = \alpha_i = \frac{1 - \beta}{1 - \beta k_i}$$

$^{11}$ The inverse class frequency of a class $i$ is defined as $\frac{dataset \ size}{occurrence \ of \ class \ i}$. 

2.5.3 Data Augmentation

2.5.3.1 Textual Data Augmentation

Instead of re-weighting samples, augmented samples from underrepresented classes can be generated. This concept of augmentation is widely applied to image processing tasks. However, a big problem of augmenting textual data consists of verifying whether the true label of the data is preserved by augmenting it. For example, one could argue that the specific order of words is a major characteristic of the speaking style of an author and thus, a sentence modified like this should no longer be attributed to said author.

A fairly popular text augmentation technique called EDA\textsuperscript{12} has shown to be especially effective for datasets with little samples[135]. It suggests to perform four different kinds of augmentations in a sentence: synonym replacement, random insertion, random swap and random deletion of a word. In the papers’ studies the authors argue that augmented samples remain close to the original samples in the latent space, suggesting that they actually maintain their true class label. The algorithm SMOTE\textsuperscript{13} yields a similar pathway in manifold aspects[17].

Another textual data augmentation technique is called back-translation which translates sentences from one language to another and back[115]. This way, the overall meaning of the sentence is preserved, while sentence composition, word choice etc. could vary. For the task of this thesis, I will not elaborate on back-translation, since it logically can not guarantee the preservation of the labels of the individual authors.

Other augmentations such as the removal of subordinate clauses or negating whole sentences to use them as negative samples mostly require higher knowledge of the sentence structure and are in general hard to formalize properly.

2.5.3.2 Latent Space Data Augmentation

An augmentation technique specifically designed for deep learning problems is called latent space augmentation. Instead of augmenting the input samples, this technique augments data at an intermediate state often referred to as latent representation. A popular technique, called Manifold Mixup, applies this concept to two data samples in a random layer of the network[128]. Its idea stems from the paper mixup: Beyond Empirical Risk Minimization, which directly applies the principle to the raw input[147]. In a mixup step, two samples $x_1$ and $x_2$ with their corresponding label vectors $y_1$ and $y_2$ are combined in the following manner using the hyperparameter $\alpha \in [0, \infty)$\textsuperscript{14}:

\[
\begin{align*}
x_{\text{mix}} &= \lambda * x_1 + (1 - \lambda) * x_2 \\
y_{\text{mix}} &= \lambda * y_1 + (1 - \lambda) * y_2 \\
\lambda &\sim \text{Beta}(\alpha, \alpha)
\end{align*}
\]

Instead of adjusting $y$, the two cross-entropy losses of both original labels $y_1$ and $y_1$ can be combined and weighted accordingly, which is considered to be more robust:

\[
L_{CE-Mix}(y_{\text{pred}}, y_1, y_2) = \lambda * L_{CE}(y_{\text{pred}}, y_1) + (1 - \lambda) * L_{CE}(y_{\text{pred}}, y_2)
\]

where $y_{\text{pred}}$ are the logit values of the last layer of the network.

\textsuperscript{12}Abbreviated from Easy Data Augmentation.

\textsuperscript{13}Abbreviated from Synthetic Minority Oversampling Technique.

\textsuperscript{14}The hyperparameter $\alpha$ controls the probability distribution of mixing coefficient $\lambda$, where higher values for $\alpha$ increase the likelihood of having a 50/50 mix of the two samples (bell-shaped) and a value below zero pushes $\lambda$ towards zero and one (inverted bell-shaped). If $\alpha$ is equal to one, the distribution is uniform.
2.5.4 Multi-Class to Binary / Ensemble Learning

Since different classes usually require different model complexity, training strategies and hyperparameters, one pathway to tackle imbalanced data is to focus on one class at a time and then combine the knowledge learned. The problem is therefore re-formulated into multiple binary problems, where each model learns to predict whether a sample belongs to a specific class or not. This ‘ensemble’ of specialized models and their predictions can then further be processed to jointly predict samples of all classes. Usually, this is achieved by selecting the label of the classifier which predicts the highest score/probability for a sample to belong to the class it has specialized on. In the context of deep learning, this concept has e.g. been applied to image classification tasks [58]. One downside of this approach is the additional overhead of the need to train one model per class and run a sample through all models to get a prediction. This complicates the deployment of the algorithm.

2.5.5 Common Regularization Techniques

In addition the methods mentioned above, general regularization techniques can furtherly be applied. These techniques include all tweaks which help a network generalize better to novel data and be more robust overall. A popular technique is the L2 regularization\(^{15}\), which adds a normalization penalty term to the error function depending on the weights of the neurons. It is an effective method to fight overfitting\(^{16}\). Another commonly used technique is to apply dropout to fully-connected layers. With dropout, each connection inside the layer is randomly disabled with a certain probability at every iteration. This way, the capacity of the network is restricted and the network is less likely to assign “big tasks” to single neurons.

I will conduct ablations and experiments on this matter in chapter 5.

---

\(^{15}\) Also referred to as weight decay or ridge regression.

\(^{16}\) Overfitting is understood as a scenario, where the network can not generalize well to unseen data but rather memorizes the training data.
Generally speaking, there exist two kinds of related topics to this thesis: Topical parallels are compiled into section 3.1, while the relevant technical work is covered in sections 3.2 and 3.3.

### 3.1 Fake Detection

The terms *fake news* and *deepfakes* gained an impressive amount of popularity over the last few years. Fake news are broadly defined as media which have no basis in fact, but are presented as being factually accurate\[4\]. Deepfakes are understood as any sort of media generated synthetically using deep learning technologies. Faces in videos can be exchanged and voices can be emulated to generate a video of a person, in which she is displayed in any conceivable scenario.

In times of a global pandemic, harsh governmental action against freedom of speech\(^1\) and mass manipulation using big data and social networks\(^2\), the need for detecting and filtering out fake information is essential for the well-being of society.

#### 3.1.1 Fake News

Due to the enormous dissemination speed of information in social media, big tech companies have recognized the potential of user manipulation and took measures to combat the spread of false information. More than two years ago, Facebook set up a *War Room* to monitor and regularize the spread of information during critical events[95]. However, a recent study related to misinformation about the COVID-19 pandemic shows that social media companies are far from perfect at removing false information[9][102].

---

\(^1\) For example, China developed a whole arsenal of tools to spread pro-governmental information, mute voices of resistance and hide events which put the government in a bad light[105][52].

\(^2\) One famous example is the influence of the now insolvent company *Cambridge Analytica* on the 2016’s Presidential Elections. According to investigations the company spread personalized political advertisement with shady information and misstatements on Facebook[15][122].
In this context, it should be noted that a world-wide pandemic of a novel virus with diverging opinions in scientific circles may not be the best example to put the current state of affairs to the test.

On the technological side, there exist two general directions to cope with fake news: On one hand, one can try to falsify fake content by fact-checking. This can either be done by experts[133], in a crowd-sourced manner[42] or by using algorithmic models[81][146][39]. On the other hand, one can try to halt the propagation of malicious information by searching for suspicious spreading patterns (e.g. information spread by social bots[16][121][94][130]. Recent publications use graph neural networks[12] or recurrent neural networks[86] to tackle this problem.

3.1.2 Deepfakes

A study from 2019 showed that 60% of people in a survey couldn’t separate AI-generated speeches of Donald Trump from real ones[132]. In the end of 2019 the U.S. Congress introduced the DEEP FAKEES Accountability Act, with the goal to "combat the spread of disinformation through restrictions on deep-fake video alteration technology" and establish a task force to develop tools for deepfake detection[25]. Both examples reveal the necessity to act on deepfake detection immediately.

3.1.2.1 Deepfake Technology

The rise of deepfakes is largely attributable to the recent progress of Generative Adversarial Networks (GANs)[49], since such network architectures allow a model to simultaneously train a generator network, which creates the deepfakes and a discriminator network, which is trained to discriminate between real and fake output. Ever improving impressive results in image[74][117], voice[126][123][68] and video generation[118][124] have been achieved over the past years. Usually, these networks deliver better results the more data is available for a certain subject. This further endangers famous and influential people, since they typically receive a lot of media attention. Many deepfake creation algorithms are publicly available on Github[38] and Google Colab[51] or even directly integrated into social media[136] and can be operated with little additional knowledge.

Figure 3.1: A screenshot from a video deepfake algorithm animating Sean Bean’s head to mimic my facial expression and movements using a pre-trained network provided on Google Colab[51].

3.1.2.2 State-of-the-Art in Deepfake Detection

Since there exists a multitude of deepfaked content, I will focus on the two most relevant kinds for this thesis: images and video.
CHAPTER 3. RELATED WORK

Detecting fake images is mostly approached either by using forensics\[40\][131], some sort of deep learning approach such as a convolutional neural network\[101\][27][87][73], or a mixture of both\[107\] to distinguish between real and fake pictures. The detection of deepfake videos is usually approached by decomposing the video frame-by-frame and focusing on detecting face manipulations, since videos of people are the most popular kind of deepfaked video\[110\][3][65][82]. Such algorithms mostly contain a facetracker which detects the face of a person and provides the extracted face to a convolutional neural network for classification. Other methods approach the problem by using biological patterns: Li et al. focus on detecting eye blinking from videos, since, as they argue, blinking is not well reconstructed by deepfake algorithms\[138\]. Recent work from Ciftci et al. and Qi et al. focus on spatiotemporal patterns in biological signals which are extracted from the video to separate real videos from fake ones\[23][100]. Agarwal et al. focus on exploiting inconsistencies with certain spoken sounds and their corresponding mouth dynamics\[109\]. A completely different kind of approach, which is partly aligning with this thesis, is to use external sources to validate the content of the video. Hasan and Salah developed a framework which uses the videos metadata, blockchain\(^3\) and smart contracts\(^4\) to trace back the video source to verify trustworthiness\[60\].

3.1.2.3 Problems with Deepfake Detection

Nevertheless, recent history has shown that it seems much easier to generate a new deepfake algorithm or modify the faked video in a specific manner than being able to build a robust deepfake detector. Bypassing such detection algorithms can be achieved by adversarially modifying fake videos frame by frame. Neekhara et al. demonstrate successful attacks on state-of-the-art deepfake detection algorithms in white as well as black box\(^5\) environments\[92\]. Gandhi et al. follow a similar path and achieve a drop of detection accuracy of 70\% by using adversarial image perturbations\[45\]. Unfortunately, this problem is not domain-specific and has shown to be a general vulnerability of neural networks trained on classification tasks\[50][93][116\]. However, it has mostly been ignored by publications about deepfake detection so far. According to Hany Farid, a computer science professor at the University of California, and Nasir Memon, a computer science professor at the University of New York, it is only a matter of time until conventional detection algorithms will no longer be able to detect deepfakes\[134\].

3.2 Text Classification

Text classification is a subdomain of NLP and comprises the task of processing, analyzing and categorizing text into pre-defined categories. A common application is spam detection, where the text corpus consists of emails which are either classified as spam or non-spam depending on the internal structure, word usage and phrasing of the email. While the problem was approached with Support Vector Machines\[69\], Expectation Maximization\[5\] or Naive Bayes\[129\] in the beginning of 2000, state-of-the-art is nowadays achieved by using deep neural networks, removing the necessity to manually build and extract features from the text. Most architectures consist of an embedding layer, followed by an RNN\[142\], a special form of such (LSTMs\[1\], GRUs\[46\]) or a CNN\[26][140\] and some sort of fully-connected or attention layer at the end. Research shows that a combination of convolutional and recurrent neural networks (RCNNs\[6][56\]) or convolutional networks and LSTMs (C-LSTMs\[148\]) could possibly yield benefits.

\(^3\) A blockchain is a growing list of records (blocks) which are linked using cryptographic hashes. Popular fields of application are cryptocurrencies, financial services and smart contracts.

\(^4\) Generally speaking, smart contracts are a sort of transaction protocol used to verify the partners identity.

\(^5\) The terms white box here means that the detection algorithm architectures were visible for the authors, whereas black box means that the authors could only work with outputs of the algorithms.
3.2.1 Text Representation

One part of the classification pipeline, which still has to be done (partly) manually, is the discretization of text, i.e. converting text into numbers and possibly extracting features from it. There are three general directions of discretizing text\textsuperscript{6}: Word by word, character by character and Part-of-Speech tags (POS).

3.2.1.1 Word Representation

Arguably the most common approach to discretize text is by focusing on single words. The simplest method is to create a lookup table from the training corpus, where each word is mapped to a unique digit. However, all the semantic information about the words is discarded with this static method and it cannot handle words it has not seen previously.

Other approaches focus solely on the frequency of words, assigning a fixed probability to previously unseen words, such as a Weighted Word model or TF-IDF\textsuperscript{7}. Unfortunately, neither capture the word position in the text and discard most semantic meaning.

To capture semantic similarities between words, researchers introduced so-called embeddings. In an embedding, each word is represented as a one dimensional vector with \( n \) entries. Even though the concept of representing words as numerical vectors has its roots in the 1970s\textsuperscript{111}, it became state-of-the-art only after adding unsupervised training to the method. The three most popular approaches are Word2Vec\textsuperscript{90}, fastText\textsuperscript{13}\textsuperscript{70} and GloVe\textsuperscript{8}. While Word2Vec is usually trained on the given training corpus before the actual task, GloVe and fastText are mostly applied as pre-trained embeddings which have been trained on hundreds of billions of tokens\textsuperscript{9}. All three models are particularly popular because they succeed in capturing fine-grained semantic and syntactic regularities of words. However, they fail to capture polysemy and are unable to deal with words which were not part of the initial training.

3.2.1.2 Character Representation

Another approach is to create an alphabet from all symbols present in the training data and then use a lookup table to assign a number to each letter and punctuation mark. It’s main advantage is its ability to represent any possible word and that languages usually contain less than 100 symbols. On the other hand, a word dictionary for a training corpus can easily contain hundreds of thousands of words. However, a lookup table is again restricted in extracting semantic meaning from sequences of characters.

3.2.1.3 POS Representation

The last approach present in the literature is usually used to do linguistic text analysis. It maps every word to its specific part of speech such as nouns, adjectives and verbs. To achieve the correct tagging, several algorithms using neural networks\textsuperscript{64}\textsuperscript{125}, dynamic programming\textsuperscript{22} or SVMs\textsuperscript{67} have been developed. Usually, this kind of representation is only useful if the actual content of the text does not matter for the problem.

\textsuperscript{6} In technical literature this process is usually called encoding or embedding.

\textsuperscript{7} Abbreviated from Term Frequency–Inverse Document Frequency.

\textsuperscript{8} Abbreviated from Global Vectors for Word Representation.

\textsuperscript{9} Tokens are here understood parts of a sentence usually separated by white space. Therefore, words as well as e.g. emoticons are considered tokens.
CHAPTER 3. RELATED WORK

3.2.2 Authorship Attribution

The task of author attribution is a subfield of text classification whose goal it is to identify the author(s) of a text from a range of possible writers. The text can either be a sentence, a paragraph or a whole article. The methods applied mostly vary in their used text corpus, text/feature representation and classifier.

3.2.2.1 Traditional Learning Approaches

Early approaches included statistical methods using the Naive Bayes classifier[91] and mainly focused on word frequencies. Afterwards, SVMs were considered state-of-the-art for a long time, as they were able to cope with larger data and feature representations[35][112]. There exists a variety of other approaches such as meta-learning models, nearest neighbour models and hybrid approaches, which are no longer in use, neither in computer science nor in linguistics.

To get a sense for modern linguistic-centered approaches, the reader is advised to consult the work from Petr Plecháˇc, who used rhythmic types, rolling attribution[34] and other features to determine which parts of Henry VIII were actually written by Shakespeare himself[97].

3.2.2.2 Approaches using Neural Networks

Current state-of-the-art in author attribution is dominated by various neural network architectures. The two main kinds of models are CNNs and RNNs with various classification heads and different combinations of inputs.

Yoon Kim uses a stack of three convolutions with different filter sizes (visualized in figure 3.2.2.2) and the Word2Vec embedding to achieve solid results with various classification tasks[75]. Shrestha et al. apply a similar concept to short texts using character n-grams[98]. Hitschler et al. feed their convolutional network with word embeddings as well as with the corresponding POS tag which lead to a gain in classification accuracy[71]. Further, Boumber et al. extended the network from Kim to be able to handle multi-label authorship attribution and found GloVe to yield better results than Word2Vec in their setting[29]. While Hitschler et al. and Boumber et al. process whole documents as input, the other approaches work with single sentences or tweets.

Figure 3.2: A visualization of the proposed network architecture of Yoon Kim. Picture taken from [75].
However, most recent work in authorship attribution focuses on RNNs and especially LSTMs, which seem to have suitable properties for the task. Douglas Bagnall applied an RNN to a dataset of four different languages, using character n-grams which achieved the best score at the PAN 2015 Author Identification task[32]. Alsulami et al. successfully applied LSTMs and BiLSTMs to source code authorship attribution[8]. The model of Jafariakinabad et al. first learns a syntactic representation from the POS tags of the sentences and then performs classification on only these representations using an RNN[66]. LSTMs and GRUs in combination with the GloVe embedding have further been applied on this task by Gupta et al.[57]. All of these networks work with whole documents as samples.

### 3.3 Language Modeling and Text Generation

Since language models can be fine-tuned to the task of author attribution, I will introduce a few key concepts here. A language model is, generally speaking, a probability distribution over different sequences of words belonging to a certain language. The modelled language can either be domain-specific (e.g. the language of a single person) or cover a broader spectrum (e.g. English or German).

Modern language models usually pre-train on a self- or unsupervised task and are then fine-tuned to a more specific task such as classification or translation. The pre-trained weights are used as initialization of the network during fine-tuning, following the general idea of transfer learning. In order for the model to be able to form useful representations during pre-training, huge amounts of text data are necessary. This data is usually scraped from internet sources such as Wikipedia. To be able to process the vast amount of sequential input most state-of-the-art models use some sort of transformer model (introduced in section 2.4), which facilitates training these models in parallel. Googles BERT\(^{10}\), introduced in 2018, contains 340 million parameters and was trained on 3’500 millions words[30]. It set new highest benchmarks in various NLP tasks.

![Figure 3.3: A visualization of the BERT architecture for pre-training (left) and fine-tuning (right). Picture taken from [30].](image)

---

\(^{10}\) Abbreviated from Bidirectional Encoder Representations from Transformers.
Another work from Google, XLNet, eliminated several weak points of BERT while maintaining a similar architecture and outperformed it on 20 tasks\[139\]. Inspired by the success of BERT and XLNet, Microsoft released Turing NLG\[11\] in early 2020, which consists of 17 billion parameters and again pushed the boundaries of the state-of-the-art\[89\]. However, arguably the most versatile language model at the moment is OpenAI’s GPT-3\[12\]. The model consists of 175 billion parameters, which is roughly double the amount of neurons a human brain contains, and was trained on almost 500 billion tokens\[14\]. GPT-3 is able to perform a variety of NLP tasks at state-of-the-art or better without being specifically trained, setting a new milestone for general artificial intelligence.

Another pathway in this domain are generative adversarial networks. Most research combines a GAN architecture with LSTMs\[141\][145], general RNNs\[99\] or concepts from reinforcement learning\[144\][55]. These models are trained by either predicting a missing piece of a sentence\[41\], relatively ranking different outputs by a rank score\[83\] or by focusing on knowledge distillation\[59\]. This way, GANs are able to generate knowledge about general language structures and contents.

\[11\] Abbreviated from Turing Natural Language Generation.

\[12\] Abbreviated from Generative Pre-trained Transformer 3.
Deepfake Detection using NLP

As shown in section 3.1.2, deepfake detection nowadays lacks robustness, accountability and endurance. As multiple experts of the field state, there exists an expiration date for all conventional deepfake detection methods. It seems to be only a matter of time until deepfakes can no longer be distinguished from real videos on any level. Since conventional deepfake detection methods focus on images and audio and not the semantic content of the video, there is yet another level of detecting deepfakes left to be explored further.

Given a big enough labeled dataset of speech samples of people, it should be possible to train a neural network to confidently predict whether a specific person could have said a certain phrase or not. This approach entirely neglects the pathway of conventional algorithms by focusing only on language rather than artifacts and other irregularities. Such an approach is naturally limited to persons of which there exist previously recorded verifiable speech data. Since most impactful people usually receive a fair amount of media attention, this approach could nevertheless protect highly influential people and therefore yield great value in the fight against the spread of deepfakes.

After describing the problem, I will introduce the dataset I created and describe a few insights regarding the data. Subsequently, I will present my novel network *PolitBERT*, which I used to tackle the proposed problem.

4.1 The Problem

At the time I am developing this thesis, the 2020’s Presidential Elections of the USA are an ongoing topic in the media. Due to America’s hegemonic position and its global influence, these elections certainly have a tremendous impact on the world. The biggest agents in the election are (former) candidates, presidents and vice presidents. I identified these people as being exposed to the biggest potential harm from the propagation of fake information and deepfakes (an example of which is displayed in figure 4.1). As these politicians are frequently present in the media, they meet the criteria of being able to detect deepfakes with NLP. Thus, I defined the problem to be solved as being able to confidently attribute authorship of single spoken sentences to the correct American politician.

2 https://youtu.be/cQ54GDmleL0
4.2 Dataset

Having defined the area of application of the model, I searched the internet for a fitting dataset. In a best-case scenario this dataset would consist of an equal amount of real speeches and interviews as well as deepfaked content. Fortunately, I could not find any publicly available, harming deepfakes of American politicians. On platforms such as YouTube, I only found deepfakes or satiric sketches, of which the content is not particularly sensitive or authentic. Even though certain speaking patterns of the politicians were adapted in order to mock the specific person, the amount and quality of the data would be insufficient to gain reasonable insights. Therefore, I focused on real speeches, interviews and press briefings only:

- **Size**: as big as possible, in the range of hundreds of thousands sentences
- **Authors**: American politicians which hold relevant positions in the 2020’s Presidential Elections
- **Content**: Transcriptions of spoken sentences only
- **Verification**: A possibility to verify and review the data

Unfortunately, there exist very few datasets of American politicians online. Most of which either focus on tweets as well as speeches, are outdated or of small size. To increase the robustness and diversity, the data needs to include speeches as well as interviews and press briefings. It is necessary to have records of different environments and settings, since, for example, speeches could have been pre-written, while interview sentences are mostly formed spontaneously. It is furthermore important to gather transcripts from different eras, since the speaking style and patterns could have changed over time. For the rest of this chapter, I will use the term ‘speech’ as a general term for interviews, press briefings and actual speeches.
4.2.1 Creation of the Dataset

Since neither of the datasets I found met the desired requirements, I built a novel dataset by scraping\(^3\) transcripts off the internet. I chose to gather data of the following American politicians due to their major roles in the 2020’s Presidential Elections:

- **Donald Trump**: Former President of the United States from 2016 to 2020 and presidential candidate of the Republican party in 2020
- **Joe Biden**: Presidential candidate of the Democratic party in 2020
- **Mike Pence**: Vice President of the United States from 2016 to 2020
- **Kamala Harris**: Senator of California and Vice Presidential Candidate of Joe Biden in 2020
- **Barack Obama**: Former President of the United States from 2008 to 2016
- **Hillary Clinton**: Former presidential candidate of the Democratic party in the 2016 election
- **Bernie Sanders**: Second presidential candidate of the Democratic party, who quit the Election Campaign in March 2020

This set of people is fairly versatile, since it consists of politicians holding various positions and a variety of language styles among them. Apart from the main seven politicians, I furtherly gathered transcripts of various other English-speaking politicians to provide data for potential unsupervised pre-training. I will refer to the set of these seven politicians as \(\text{Set}_7\) and to the set of all scrapings as \(\text{Set}_{\text{all}}\).

4.2.1.1 Included Websites

The American company Rev provides an immense amount of transcripts of various politicians for free on their website. For each selected politician I found transcripts of the vast majority of their latest media appearances\(^4\). However, speeches older than three to four years are mostly not transcribed by Rev. Therefore I included the website American Rhetoric, which transcribed a few hundred speeches and interviews of Barack Obama over the time from 2002-2017\(^5\). The website factba.se is arguably the biggest database of transcripts of Donald Trump, heading back to 1976 with over 3000 transcripts of speeches and interviews\(^6\). Another website called The American Presidency Project transcribed almost 200 speeches and interviews of Hillary Clinton over the time from 2007-2016\(^7\).

For the politicians which are not part of the main dataset, I used the following sources:

The website The Grammar Lab collected speeches of all American presidents up to Barack Obama\(^8\). Using the database from Rev, I scraped speeches and interviews of five more rather famous politicians from the United States. To include more diversity, I furthermore collected data from other countries. Justin Trudeau, the prime minister of Canada, has his own website, where transcripts of almost 200 of his speeches are provided\(^9\). The UK has their own political speech archive, from which I collected over 4000 speeches of various British politicians\(^10\).

\(^{3}\) (Web) Scraping is a common term which describes extracting data (texts, images etc.) from websites.

\(^{4}\) https://www.rev.com/blog/transcript-category/political-transcripts

\(^{5}\) https://www.americanrhetoric.com/barackobamaspeeches.htm

\(^{6}\) https://factba.se/transcripts


\(^{8}\) http://www.thegrammarlab.com/?nor-portfolio_cat=corpora

\(^{9}\) https://pm.gc.ca/en/news/speeches

\(^{10}\) http://www.ukpol.co.uk/speeches/
4.2.1.2 Webscraper

Since all websites are coded with different HTML-tags and layouts, I wrote a scraping tool for each of the websites mentioned above. To simplify the process, I used the library Selenium to browse through the different pages and the library BeautifulSoup to extract the content. Having scraped the transcripts, I applied the following processing:

- Verification of author: Text is only included, if it begins with "Author: ..." or if it is a whole speech of said person. Questions from Interviewers or parts of other people talking are removed.
- Removal of sentences, which include inaudible words or cross-talk to only include text without interference.
- Removal of tab stops, new lines and other spaces (e.g. \xa0)
- Division of the transcript into paragraphs: A paragraph can be seen as multiple logically connected sentences (e.g. a whole answer to a question). Paragraphs are then separated by a new line and concatenated to one transcript string per speech.
- Extraction of the URL, as well as the date and title of the speech. This information can be used to verify and trace back the content of the dataset.

I used regular expressions to filter for non-author sentences and other noise. The most important expressions and the general structure of the webscraper can be found in appendix A.1. All transcripts are stored as dataframes using the library Pandas for each author, which is a common tool to browse and store textual data. The whole process is visualized in figure 4.2.

Figure 4.2: An example of the common format of a transcript and how it is processed by the webscraper.
4.2.1.3 Transcription Quality and Drawbacks

To verify the transcription quality, I randomly chose five speeches/interviews from each website, looked up the video and compared the transcript with the original audio. All transcripts have been of great quality, correctly transcribing each understandable word mostly with correct grammar and punctuation. Nevertheless, there are a few minor drawbacks:

Even if the video and audio quality is good, it can happen that a certain word is not understood at all. The transcripts mark such sections with tags such as [inaudible] or (laughter). As mentioned above, such sentences have been removed from the data.

Another remark is that the websites transcribe "thinking pauses" and hesitations in different manners. For example, Rev mostly transcribes these as '...', while other websites transcribe it as '–'. The actual sentence is only sometimes continued after such a pause. On the other hand, '–' is also used to mark an insert (e.g. "Yesterday – Tuesday – I outlined..."), so these pauses cannot be removed. Therefore, this ambiguity can not be resolved by looking at the context of the sentence and shows one limitation of processing content from different websites and transcribers.

4.2.1.4 Increasing Robustness and Decreasing Ambiguity

To cope with the limitation of varying transcription styles all chosen politicians have at least a few speeches which were scraped from Rev. If there exists overlap between different sources (the same speech was transcribed by multiple sources), the texts from Rev are preferred to keep the transcription consistency as high as possible. To catch non-trivial copies (recorded on a different date and/or with another/misleading title from two different websites), I calculated the similarities between all speeches from one author and then manually discarded duplicated speeches. The two applied measures are the Hamming distance\(^{11}\) (to include an editing-based approach) and the Jaccard similarity\(^{12}\) (to include a token-based approach), which are two of the most common pathways to measure text similarity.

4.2.2 Dataset Analysis

As mentioned earlier, I created two main datasets: \(\text{Set}_7\) includes the seven chosen American politicians, for which I want to build a classifier and \(\text{Set}_{all}\) consists of speeches from various English speaking politicians, which is used for pre-training tasks. Since \(\text{Set}_7\) plays the central role in this thesis, I will only analyze this dataset in-depth.

4.2.2.1 Dataset of all Politicians

\(\text{Set}_{all}\) consists of speeches and interviews from 1091 different politicians and three different countries (America, United Kingdom and Canada) gathered from 1789 until 2020. It has the following properties:

- Amount of speeches: 9’659
- Amount of sentences: 1,46 M
- Amount of words: 21,71 M
- Amount of unique words: 70’375

\(^{11}\) The Hamming distance counts in how many places two speeches are different, meaning the amount of substitutions needed to form one text from the other. If the metric yields a lower number, the texts are more similar on a structural level.

\(^{12}\) The Jaccard similarity is the coefficient given by the division of the intersection of two texts by the union of them. If the metric yields a number close to 1, the texts contain many common words, while not containing as much uncommon words, which can be interpreted as a form of similarity.
4.2.2.2 Dataset of the Politicians in Focus

Since it is crucial to be familiar with the used data and its properties to successfully apply a classification algorithm, I will analyse Set \( T \) more thoroughly. The basic properties of the dataset are:

- Amount of speeches: 4’201
- Amount of sentences: 0.95 M
- Amount of words: 11.25 M
- Amount of unique words: 40’151

![Figure 4.3: The distribution of sentences per author in Set \( T \).](image1)

![Figure 4.4: The distribution of sentence lengths in Set \( T \).](image2)
Even though I tried to scrape as many speeches for each individual as possible, the dataset is heavily imbalanced as shown in figure 4.3. The ratio between the most and the least frequently present authors is approximately 1:200. This can be explained due to the fact that Donald Trump has been present in the media for 40 years, whereas other politicians, such as Kamala Harris, only recently gained enough attention for their speeches to be transcribed. Nevertheless, the amount of data should be sufficient to verify whether a certain technique is promising or not.

85% of the sentences consist of 1-20 words distributing according to the power law and its 80-20 rule\(^\text{13}\) (displayed in figure 4.4). This fact can provide clues about which lengths of sentences a model should be able to deal with.

<table>
<thead>
<tr>
<th>Author</th>
<th># Speeches</th>
<th>AVG WL</th>
<th>AVG SL</th>
<th>LD</th>
<th>Top three words after SW removal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Donald Trump</td>
<td>3225</td>
<td>4.03</td>
<td>12.14</td>
<td>0.0034</td>
<td>going, people, know</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>464</td>
<td>4.43</td>
<td>22.20</td>
<td>0.0177</td>
<td>people, us, going</td>
</tr>
<tr>
<td>Hillary Clinton</td>
<td>200</td>
<td>4.24</td>
<td>18.62</td>
<td>0.0234</td>
<td>people, know, going</td>
</tr>
<tr>
<td>Joe Biden</td>
<td>184</td>
<td>4.10</td>
<td>15.17</td>
<td>0.0266</td>
<td>going, people, get</td>
</tr>
<tr>
<td>Kamala Harris</td>
<td>63</td>
<td>4.23</td>
<td>19.02</td>
<td>0.0702</td>
<td>people, us, know</td>
</tr>
<tr>
<td>Bernie Sanders</td>
<td>39</td>
<td>4.35</td>
<td>18.27</td>
<td>0.0505</td>
<td>people, country, going</td>
</tr>
<tr>
<td>Mike Pence</td>
<td>22</td>
<td>4.39</td>
<td>21.49</td>
<td>0.0675</td>
<td>president, people, going</td>
</tr>
</tbody>
</table>

Table 4.1: Statistics of the seven authors in $Set_7$. AVG: average, WL: word length (in characters), SL: sentence length (in words), LD: lexical diversity (unique words divided by all words), SW: stop-words.

Table 4.1 displays several key figures of the different authors. Donald Trump’s lexical diversity is significantly smaller compared to the lexical diversity of the other politicians. This is easily explained by the fact that the amount of data available for Donald Trump is significantly higher as well. However, Donald Trump’s average sentence and word length is likewise significantly shorter than it is the case for all other politicians. This seems to correlate with his characteristic simple and short syntax. Even though many speeches and interviews of Barack Obama are included, his average word and sentence lengths are the highest in the dataset; correlating with the elaborate and accurate language he is known for. In terms of vocabulary usage, each person includes *people* in her three most used words, signalling closeness to and care for the public.

### 4.3 PolitBERT

Recent advances in NLP have shown that massive pre-trained language models, fine-tuned on a specific task, yield state-of-the-art results in various tasks such as language translation or question answering. For this reason, I decided to adapt such a model to the proposed task. As presented in section 3.3, there currently exist four popular language models: BERT, XLNet, Turing NLP and GPT-3. For my experiments, I chose to focus on the base version of BERT\(^\text{14}\), since it proved to be a highly successful model in research of various fields, manageable in terms of model size while yielding comparable results to larger models. This architecture has recently been transferred to computer vision tasks successfully\(^\text{31}\).

\(^\text{13}\) The 80-20 rule is also known as the pareto principle, where 80% of consequences come from 20% of the causes.

\(^\text{14}\) The authors presented two versions of BERT: BERT\(_{BASE}\), consisting of 12 transformer blocks and attention heads and a hidden dimensionality of 768 and BERT\(_{LARGE}\), consisting of 24 transformer blocks and attention heads and a hidden dimensionality of 1024.
4.3.1 The Network

BERT’s network architecture is identical to the original Transformer described in section 2.4, extended with additional transformer blocks. Its capabilities mainly stem from pre-training on a huge dataset with unique training objectives. However, there are a few special tools used to achieve successful pre-training: Instead of ReLU, the authors use the more smooth GELU activation function for hidden units\[^{15}\]. Instead of a usual pre-trained word embedding (e.g. GloVe), they train a WordPiece embedding on the data\[^{137}\]. The idea of this embedding is that infrequent words are split into "wordpieces". Even though the vocabulary is limited, all previously unseen words can then be built as a combination of wordpieces. For example \textit{astrological} is represented as \textit{astro## + ##logical}. This embedding can deal with previously unseen words but maintains information about the usage of frequent words by not only focusing on an alphabet.

Using BERT, there are two main pathways which have proven to yield promising results in tasks similar to author attribution:

1. Take the original pre-trained weights of BERT, add a fully-connected classification head and then fine-tune all parameters to the given task.

2. Pre-train the model by using domain data and the proposed pre-training tasks and then fine-tune all parameters to the given task.

While the first method can be applied quite straightforwardly, the second one requires preceding training.

4.3.2 Pre-Training

Originally, BERT was pre-trained on two tasks:

1. Masked Language Modeling (MLM): In each sentence 15% of all wordpieces are randomly masked. They are either blanked (80% chance), replaced (10% chance) or left the same (10% chance). Given this modified sentence, the model is trained to predict the masked wordpiece and therefore learn sentence structures and fitting words.

2. Next Sentence Prediction (NSP): The model is given two sentences. In 50% of the time the second sentence is actually the successive sentence of the previous one. Here, the training objective is to determine whether the second sentence actually is the correct successor. Like this, the model learns to understand the relationship between the sentences.

Following these objectives, researchers specialized BERT on various domains by pre-training on specific corpora, which increased the performance on domain-specific tasks. This concept has been applied to semantics (LIBERT\[^{79}\]), linguistics (VL-BERT\[^{119}\]), biomedicine (BioBERT\[^{80}\], PubMedBERT\[^{53}\]), clinical medicine (ClinicalBERT\[^{63}\]) and scientific publications (SciBERT\[^{10}\]).

4.3.2.1 Implementation

Similarly to these approaches, I trained BERT on MLM using Set\(_{all}\) after removing a validation set to assure non-biased fine-tuning. The weights of the network were initialized with the original pre-trained weights presented in the paper to include general language knowledge. I kept most hyperparameters identical to the proposed values of the original paper and used a linearly decreasing learning rate starting from $5 \times 10^{-5}$.

\[^{15}\] Abbreviated from Gaussian Error Linear Unit.
This learning rate is lower than the one proposed in the paper (1e−4), but since the weights are initialized with the original pre-trained values it makes more sense to start at a lower learning rate to benefit from the implicit knowledge already present. As a vocabulary, I used the same WordPieces which were used for the initial BERT training, since I’m starting with the exact same model and there doesn’t exist a huge domain-specific vocabulary as for, e.g., medicine.

### 4.3.2.2 Results

These are a few examples of the pre-training, where the masked word is labeled as [MASK] and a confidence score in [0,1] indicates how well a word fits according to the model:

- Barack [MASK] was the 44th president of the USA.  
  *Obama* - 0.999, *Hussein*\(^{16}\) - 0.0002  
- Trump got 200 electoral [MASK].  
  *votes* - 0.652, *points* - 0.054  
- Make [MASK] great again!  
  *America* - 0.981, *it* - 0.005  
- [MASK] Bernie Sanders.  
  *Crazy* - 0.356, *Senator* - 0.044  
- I’m sure Hillary Clinton [MASK] the election.  
  *lost* - 0.734, *won* - 0.1872

The network seems to be specialized in the implicit knowledge of the training samples. For example it correctly predicts that Bernie Sanders is a senator and that Hillary Clinton lost an election. More examples and the loss and learning rate curves can be found in appendix A.2.

### 4.3.3 Fine-Tuning

As proposed by the original paper, fine-tuning on a classification task can be realized by adding a fully-connected layer to the output of BERT. The authors suggest to conduct a grid-search over the following hyperparameter values:

- Batch size: 16, 32
- Learning rate: 5e-5, 3e-5, 2e-5
- Number of epochs: 2, 3, 4

However, I will further elaborate on fine-tuning and adapting the model to author attribution in chapter 5.

For the remaining part of this thesis, I will refer to the model with the original pre-trained weights and no further pre-training as BERT and to the model which initialized with the original pre-trained weights and then pre-trained on Set\(_{all}\) as PolitBERT.

\(^{16}\) Hussein is the second last name of Barack Obama.
For the conduction of my experiments, I focused on two main pathways:

1. **Single-class single-label classification (one vs. all):** All sentences of one politician are labeled positively, while all sentences of the other politicians are labeled negatively. The algorithms are thus trained to predict whether a certain sentence was said by this particular politician or not. This way, a model has to be trained for every politician individually.

2. **Multi-class single-label classification (all vs. all):** Each of the politicians introduced in the last chapter is labeled uniquely. The algorithms are trained to jointly predict which sentences belong to which politician. After the training, the same model can be applied to classify each politician.

I chose these two tasks since they offer diverse opportunities and pitfalls. For the first case, the data is highly unbalanced in percentage, but consists of only two classes. The overall difficulty to fit the data is lower than in a multi-class problem, since the algorithms have to determine only whether a sample belongs to an author or not. Nevertheless, optimizing networks and finding hyperparameters seven times, rather than only once, creates a non-negligible overhead and reduces the ease of application.

For the second case, the biggest challenge is to handle the heavily imbalanced dataset of seven classes, i.e. not to overfit on certain authors while nevertheless providing enough complexity to jointly learn patterns of each author.

In the remainder of this chapter, I will introduce the general pre-processing steps, the metrics applied and the baseline models in sections 5.1-5.3, then discuss the performance of the individual models in detail in sections 5.4-5.6 by providing benchmarks, qualitative results and ablations.
CHAPTER 5. EXPERIMENTS

5.1 Pre-Processing

I applied the following general pre-processing to the dataset in order to remove serious ambiguities and keep the evaluation as unbiased as possible:

- All speeches are converted to lowercase letters to remove accidental CapsLocks and grammar differences.
- All speeches are divided into single grammatical sentences, where one sentence represents one sample labeled with the corresponding author.
- The networks are trained only on sentences which consist of four or more words, not counting punctuation as single words. Usually, sentences with less than four words do not yield enough information and contain a huge amount of ambiguity (e.g. "Thank you.", "That is true." or "Yes."). It is important to note that the amount of words in a sentence may differ from the amount of entries a network uses to represent it. For example, some networks and embeddings may assign individual symbols to punctuation while others may completely ignore them or split words into several separate entries.
- All identical sentences said by different authors are removed from the dataset. These are mostly cliche phrases or other general sayings such as "No, I have not." or "It was the right thing to do.". If a person said the same sentences multiple times, all occurrences are kept in the dataset (e.g. "We will make America great again!").
- After these pre-processing steps, the remaining data is divided into a fixed stratified dataset split of 85%-10%-5% (train-val-test). Every model trains on the same training data and evaluates on the same validation and testing data. No sentences of specific authors are discarded.

5.2 Metrics

To measure the performance of my models, I focused mainly on unweighted averaged accuracy for the single-class task and macro-averaged accuracy for the multi-class task, since the amount of samples is not equally distributed between classes in either of the cases. In order to make sure that under-represented classes do not have less impact on the overall performance, the accuracies are averaged.

\[
\text{Acc}_{\text{avg}} = \frac{\text{Accuracy of author} + \text{Accuracy of non-author}}{2}
\]

\[
\text{Acc}_{\text{macro}} = \frac{\sum \text{Accuracy of author}}{\# \text{authors}}
\]

However, accuracy can only tell so much about the performance of an algorithm, since it is mainly focused on correctly classified samples. Two equally important measures, often used in practice, are precision and recall:

\[
P_{\text{author}} = \frac{\# \text{sentences correctly predicted as author sentences}}{\# \text{sentences predicted as author sentences}} = \frac{\text{True Pos.}}{\text{True Pos.} + \text{False Pos.}}
\]

---

1 The term 'stratified' means that every author's data is split with the same ratio instead of splitting the data by absolute occurrences.
2 Even though it might be intuitive to discard speeches older than 10 or 20 years, it would have only affected 1-3% percent of all samples in the dataset, making it fairly negligible.
3 Note that these measures may likely differ from the accuracy over the whole dataset.
CHAPTER 5. EXPERIMENTS

$R_{\text{author}} = \frac{\# \text{sentences correctly predicted as author sentences}}{\# \text{sentences of author}} = \frac{\text{True Pos.}}{\text{True Pos.} + \text{False Neg.}}$

These two metrics can be combined to the F1-Score, which provides a sense of how sensitive and specific an algorithm is:

$$F_1 = \frac{2 \times P \times R}{P + R}$$

Since we are dealing with imbalanced multi-author data, I applied the F1-Score with macro-averaging. This variant calculates the F1-Score for each class separately and then averages them with equal weight for each class. This score lies in $[0,1]$, where higher values correspond to better performance.

The last metric applied is the ROC AUC score, which measures the area under the ROC curve. The curve is created by plotting the true positive rate against the false positive rate at different threshold settings between zero and one. A perfect algorithm would therefore achieve an ROC AUC score of one as it would fill up the whole square spanned by the curve. The main advantage over the F1-Score is that the ROC AUC score does not only work with "hard" predictions (one predicted class label per sample), but also with a probability vector which assigns a probability to each class per sample. Since neural networks usually output such a vector before returning the maximum value as the class label, ROC AUC can identify more subtle performance differences. Again, since we are dealing with imbalanced data, macro-averaging is used to counter the imbalance. The score is calculated for each class individually and then averaged without weighting.

All benchmarks presented in the following sections are the results of the models’ performance on the test set.

5.3 Baseline

As a baseline for both tasks, I used two models, one neural network and one non-neural approach, which both set new milestones for text classification. On one hand, I fine-tuned an SVM combined with TF-IDF as word representation. This combination of classifier and feature representation was often used before the introduction of neural networks to NLP tasks.

On the other hand, I implemented the three-layer CNN of Yoon Kim (introduced in section 3.2.2.2), which had significant impact on subsequent publications. I fine-tuned on all configurations mentioned in the paper and found empirically that the GloVe embedding generally yielded better results than the originally used Word2Vec embedding. Words which are not part of the GloVe vocabulary were initialized randomly.

To better examine the performance of PolitBERT, the original BERT model without domain-specific pre-training is benchmarked along the other two baseline models.

In addition to the baseline models, I conducted further experiments on various sequence processing models, namely variants of RNNs and LSTMs. Since these models did not yield results that were significantly different from the models presented here, they are not included in the main part of this thesis. Nevertheless, their benchmarks on the (practically more relevant) multi-class single-label task can be found in appendix A.4.

---

4 Abbreviated from Area Under the Receiver Operating Characteristic Curve.
5 True Pos. + False Neg.
6 True Neg. + False Pos.
7 Abbreviated from Recurrent Neural Network.
8 Abbreviated from Long Short-Term Memory.
5.4 Single-Class Single-Label

Since the best hyperparameter settings varied for the individual algorithms and authors, listing them here would be too complex. Therefore, the hyperparameter values yielding the best performance per author and model are merely documented in appendix A.3. It is important to note that in order to draw proper conclusions, I trained both BERT variants on the same hyperparameter settings. However, since the parameters were chosen according to the best performance of PolitBERT, it is possible that the performance of BERT could be slightly improved with different hyperparameter settings.

5.4.1 Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc\textsubscript{macro}</th>
<th>F1\textsubscript{macro}</th>
<th>ROC AUC</th>
<th># Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>50%</td>
<td>0.1267</td>
<td>0.5</td>
<td>-</td>
</tr>
<tr>
<td>Kim-CNN</td>
<td>84.18%</td>
<td>0.6675</td>
<td>0.9299</td>
<td>12 M</td>
</tr>
<tr>
<td>BERT</td>
<td>87.77%</td>
<td>0.7161</td>
<td>0.9616</td>
<td>109 M</td>
</tr>
<tr>
<td>PolitBERT</td>
<td>89.44%</td>
<td>0.7185</td>
<td>0.9704</td>
<td>109 M</td>
</tr>
</tbody>
</table>

Table 5.1: The benchmarks of the different models trained on the single-class task and the amount of parameters of a single model. For each model type, the metrics were macro-averaged over all seven models.
<table>
<thead>
<tr>
<th>Model</th>
<th>Clinton</th>
<th>Obama</th>
<th>Pence</th>
<th>Biden</th>
<th>Sanders</th>
<th>Trump</th>
<th>Harris</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>Kim-CNN</td>
<td>82.79%</td>
<td>87.60%</td>
<td>84.98%</td>
<td>83.01%</td>
<td>80.63%</td>
<td>87.99%</td>
<td>82.26%</td>
</tr>
<tr>
<td></td>
<td>78.90%</td>
<td>86.68%</td>
<td>73.66%</td>
<td>75.56%</td>
<td>64.10%</td>
<td>92.11%</td>
<td>67.64%</td>
</tr>
<tr>
<td>BERT</td>
<td>85.08%</td>
<td>89.51%</td>
<td>88.07%</td>
<td>87.43%</td>
<td>86.90%</td>
<td>90.41%</td>
<td>86.99%</td>
</tr>
<tr>
<td></td>
<td>73.48%</td>
<td>96.69%</td>
<td>83.75%</td>
<td>90.15%</td>
<td>77.35%</td>
<td>95.54%</td>
<td>95.73%</td>
</tr>
<tr>
<td>PolitBERT</td>
<td>86.78%</td>
<td>90.19%</td>
<td>91.91%</td>
<td>89.90%</td>
<td>88.48%</td>
<td>91.51%</td>
<td>87.33%</td>
</tr>
<tr>
<td></td>
<td>77.80%</td>
<td>95.75%</td>
<td>85.84%</td>
<td>85.58%</td>
<td>80.77%</td>
<td>95.43%</td>
<td>78.55%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>95.04%</td>
<td>94.21%</td>
<td>96.18%</td>
<td>87.60%</td>
<td>96.11%</td>
</tr>
</tbody>
</table>

Table 5.2: The accuracies of the models split up by author. Per row the top number corresponds to the unweighted averaged accuracy ($A_{cc_{avg}}$), the left digit to the author and the right digit to the non-author accuracy.
5.4.2 Discussion

Unfortunately, the SVM couldn’t model any sort of meaningful separation of the two classes independently of the choice of hyperparameters and author. It either classifies all samples as author samples or vice versa. It seems clear that an SVM is a model too simple for the proposed task and data.

Kim-CNN, on the other hand, was able to achieve fairly decent accuracies independently of the amount of author samples. However, for the most part, it was only able to achieve good generalization for non-author samples but not necessarily for author samples. The significant amount of false negatives (author samples wrongly classified as non-author samples) leads to inferior performance in the F1 and ROC AUC score. An interesting property worth of further investigation is the fact that the lowest author accuracies have been achieved for the three authors which have the least amount of samples in the dataset (Mike Pence, Bernie Sanders, Kamala Harris). It seems that the data is not sufficiently varied to generalize properly given the models’ complexity. Another explanation could be the over-sampling applied during the training. This way, sentences from authors with less samples are processed more times per epoch than samples from authors with more samples, leading to a greater impact on the network’s weight adjustments. However, since under-sampling yielded better performance for Mike Pence, I highly doubt that the sampling strategy explains the behavior.

Both BERT variants perform similarly in terms of overall performance. Generally, the author accuracy (true positive rate) is a fair bit higher for most authors with PolitBERT than with BERT, indicating that the model learned some sort of implicit knowledge of the individual politicians. Nevertheless, both models fail to perfectly distinguish author samples from non-author samples: the non-author accuracy is higher than the author accuracy for all authors except Donald Trump. This result can be (at least partly) explained with the sheer amount of data of Trump that is available online and is consistent over all models examined.

5.5 Multi-Class Single-Label

As before, to keep the main part of this thesis concise, the best-performing hyperparameter settings are documented in appendix A.5.

5.5.1 Ensembles

Aside from modifying the models to train on multi-class classification by adjusting the amount of output neurons and the loss function, I combined the binary classifiers of each author to perform the same task. This way, each pre-trained classifier first predicts the probability of a sample belonging to the author it is specialized on. Afterwards, the sample is classified as the label with the highest class probability. This procedure is commonly known as Bagging. In this case, no additional training is required, since the results are simply combined. Another ensemble I examined is the best-performing fine-tuned PolitBERT (trained on the multi-class task) combined with another PolitBERT model, which performed better on classes for which the first model performed badly. I will refer to this model as PolitBERT_{Multi-Ensemble}.

---

This is somewhat aligning to the concept of Boosting, which has been introduced by Robert E. Schapire[113].
5.5.2 Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Clinton</th>
<th>Obama</th>
<th>Pence</th>
<th>Biden</th>
<th>Sanders</th>
<th>Trump</th>
<th>Harris</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>00.00%</td>
<td>00.00%</td>
<td>88.17%</td>
<td>00.00%</td>
<td>90.60%</td>
<td>00.00%</td>
<td>68.73%</td>
</tr>
<tr>
<td>Kim-CNN</td>
<td>58.41%</td>
<td>70.47%</td>
<td>70.43%</td>
<td>64.00%</td>
<td>52.99%</td>
<td>84.77%</td>
<td>57.82%</td>
</tr>
<tr>
<td>Kim-CNN\textsubscript{Single-Ensemble}</td>
<td>53.60%</td>
<td>71.00%</td>
<td>52.69%</td>
<td>59.75%</td>
<td>51.71%</td>
<td>89.33%</td>
<td>55.64%</td>
</tr>
<tr>
<td>BERT</td>
<td>68.17%</td>
<td>74.50%</td>
<td>77.96%</td>
<td>78.83%</td>
<td>63.68%</td>
<td>85.63%</td>
<td>62.55%</td>
</tr>
<tr>
<td>BERT\textsubscript{Single-Ensemble}</td>
<td>63.60%</td>
<td>75.85%</td>
<td>69.35%</td>
<td>63.65%</td>
<td>64.10%</td>
<td>93.95%</td>
<td>55.27%</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{Single-Ensemble}</td>
<td>69.21%</td>
<td>78.47%</td>
<td>77.42%</td>
<td>77.65%</td>
<td>66.24%</td>
<td>84.64%</td>
<td>66.18%</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{Multi-Ensemble}</td>
<td>62.32%</td>
<td>75.85%</td>
<td>75.81%</td>
<td>65.74%</td>
<td>73.50%</td>
<td>93.24%</td>
<td>56.36%</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{Multi-Ensemble}</td>
<td>68.78%</td>
<td>78.75%</td>
<td>78.49%</td>
<td>76.32%</td>
<td>67.95%</td>
<td>86.65%</td>
<td>65.45%</td>
</tr>
</tbody>
</table>

Table 5.3: The author accuracies of the individual multi-class models.

<table>
<thead>
<tr>
<th>Model</th>
<th>(\text{Acc}_{\text{macro}})</th>
<th>(\text{F1}_{\text{macro}})</th>
<th>(\text{ROC AUC})</th>
<th># Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>35.36%</td>
<td>0.0123</td>
<td>0.7317</td>
<td>-</td>
</tr>
<tr>
<td>Kim-CNN</td>
<td>65.56%</td>
<td>0.5168</td>
<td>0.9205</td>
<td>12 M</td>
</tr>
<tr>
<td>Kim-CNN\textsubscript{Ensemble}</td>
<td>61.96%</td>
<td>0.5084</td>
<td>0.9071</td>
<td>84 M</td>
</tr>
<tr>
<td>BERT</td>
<td>73.05%</td>
<td>0.5748</td>
<td>0.9497</td>
<td>109 M</td>
</tr>
<tr>
<td>BERT\textsubscript{Single-Ensemble}</td>
<td>69.40%</td>
<td>0.5853</td>
<td>0.9369</td>
<td>763 M</td>
</tr>
<tr>
<td>PolitBERT</td>
<td>74.26%</td>
<td>0.5528</td>
<td>0.9509</td>
<td>109 M</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{Single-Ensemble}</td>
<td>71.83%</td>
<td>0.5814</td>
<td>0.9539</td>
<td>763 M</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{Multi-Ensemble}</td>
<td>74.63%</td>
<td>0.5615</td>
<td>0.9551</td>
<td>218 M</td>
</tr>
</tbody>
</table>

Table 5.4: The macro-averaged benchmarks of the different models trained on the multi-class task and their amount of parameters. Note that none of the ensemble models have any trainable parameters.

5.5.3 Discussion

With most configurations, the SVM was completely unable to model any sort of meaningful class boundaries. The best-performing configuration could correctly predict three authors, but was unable to predict anything for the rest of the authors. Due to sample weighting, it seems that authors with less samples are preferred since all three authors with little samples are mostly classified correctly. Since a weighting according to inverse class frequency is the most common approach, I did not conduct further experimentation on this matter for the baseline model. Because four out of seven authors are never predicted correctly, the F1 and ROC AUC scores both yield poor results.

Even though Kim-CNN seems to be able to grasp some perks of the individual authors’ language, it is only able to generalize fairly well for Donald Trump. This was to be expected due to the enormous amount of data associated with him. The results from the other authors are disappointing, however. Although Mike Pence and Kamala Harris have approximately the same amount of training samples, Pence’s speaking style seems to be more characteristic and easier to classify, since the network is performing significantly better on his data. Overall, the architecture seems to have too little complexity to identify low-level structures.
Somewhat surprisingly, Kim-CNN\textsubscript{Ensemble} is not able to perform as good as the non-ensemble version, even though the isolated single-class accuracies from the previous section are decent. This could be due to two things: On one hand, the overall loss could be too high, indicating that the classifiers have too little confidence in their predictions. Therefore, the scenario that the predicted probability of another single-author model is slightly higher appears more likely.

On the other hand, the single-class models could be too insensitive towards author samples. Most classifiers achieve high accuracies on non-author samples but not on author samples. Therefore their classifications contain a fair amount of false negatives (author samples wrongly classified as non-author samples), which could lead to misclassifications. Perhaps, this weakness could be countered by applying focal loss during single-author training, which puts more emphasis on such samples.

BERT and PolitBERT deliver comparable results, while PolitBERT overall yields better benchmarks than BERT indicating that domain-specific pre-training actually yielded some benefits. Even though PolitBERT achieves a higher overall accuracy, BERT achieves a better F1 score. Much like the previous networks, both BERT models achieve the lowest accuracy scores for Kamala Harris and the highest for Donald Trump.

Even though the single-class models of BERT and PolitBERT do not yield significantly different results, they do if their single-class models are each combined into an ensemble: The PolitBERT single-class ensemble yields better benchmarks than the multi-class PolitBERT in both, the F1 and the ROC AUC score, even though its macro-averaged accuracy is more than two percent lower. On the other hand, the BERT single-class ensemble only achieves a performance improvement on the F1-score, while sacrificing a decent amount of performance on the other metrics. Since both ensembles seem to have a great bias towards Donald Trump\footnote{The accuracy for his samples is almost 10\% higher compared to the non-ensemble models.} his F1 and ROC AUC scores might have overshadowed the other authors’ scores. Overall, the ensembles appear less balanced than the multi-class models.

Compared to PolitBERT the PolitBERT multi-class ensemble yields better performance in each benchmark performed. While the accuracy for certain individual authors drops slightly, it is improved for authors with little samples while further yielding better F1 and ROC AUC scores. The accuracies for Mike Pence and Bernie Sanders both increased by more than 1\% using this method.

5.5.4 PolitBERT Analysis

In this section, I will analyse the results of PolitBERT\textsubscript{EDA}, the best-performing non-ensemble model, more thoroughly. I chose to focus on this network, since it yields the best ability of interpretability and usability, while performing only marginally worse than the multi-class ensemble model.

5.5.4.1 Confusion Matrix

Besides the conventional confusion matrix (table 5.5), I created a heatmap of the confusion matrix, in which the tiles indicate the percentage of misclassified samples divided by author (figure 5.1).
Table 5.5: The confusion matrix of PolitBERT, where each row corresponds to the amount of samples of an author which have been classified as the author in corresponding top column. The sum of a row is therefore equal to the amount of samples of one specific author in the dataset.

<table>
<thead>
<tr>
<th></th>
<th>Clinton</th>
<th>Obama</th>
<th>Pence</th>
<th>Biden</th>
<th>Sanders</th>
<th>Trump</th>
<th>Harris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clinton</td>
<td>1135</td>
<td>222</td>
<td>10</td>
<td>84</td>
<td>35</td>
<td>136</td>
<td>18</td>
</tr>
<tr>
<td>Obama</td>
<td>230</td>
<td>2216</td>
<td>18</td>
<td>149</td>
<td>31</td>
<td>142</td>
<td>38</td>
</tr>
<tr>
<td>Pence</td>
<td>1</td>
<td>11</td>
<td>144</td>
<td>14</td>
<td>1</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Biden</td>
<td>51</td>
<td>87</td>
<td>16</td>
<td>1115</td>
<td>30</td>
<td>90</td>
<td>47</td>
</tr>
<tr>
<td>Sanders</td>
<td>13</td>
<td>11</td>
<td>2</td>
<td>30</td>
<td>155</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>Trump</td>
<td>1676</td>
<td>1566</td>
<td>267</td>
<td>1481</td>
<td>263</td>
<td>30043</td>
<td>199</td>
</tr>
<tr>
<td>Harris</td>
<td>12</td>
<td>15</td>
<td>0</td>
<td>45</td>
<td>6</td>
<td>15</td>
<td>182</td>
</tr>
</tbody>
</table>

Almost 50% of the misclassifications of Kamala Harris were attributed to Joe Biden, indicating a close relation between the two. Since most speeches of both persons have been taken from the 2020's Presidential Elections, it is highly likely that they address similar issues and were written in collaboration with the same team of consultants and writers. The same holds true for Bernie Sanders: Almost 40% of misclassifications of samples from him were attributed to Joe Biden.

A similar connection can be found with Hillary Clinton and Barack Obama. Around 40% of misclassifications of Clinton were attributed to Obama and vice versa. Even though they have no obvious close relation, both were presidential candidates of the Democratic party and stand for similar values. Therefore, the distinction between the two seems to be challenging.
5.5.4.2 Sentence Lengths

To examine whether there exists a relationship between misclassifications and sentence lengths, I divided the amount of misclassified sentences of a certain length by the amount of total sentences of this length (figure 5.2).

![Figure 5.2: The amount of misclassified sentences grouped by the amount of wordpieces they are represented with and divided by the total amount of sentences with the same length.](image)

The intuitive thought that shorter sentences are harder to classify than longer ones due to their restricted information content has only been confirmed partially. Shorter sentences are classified wrongly approximately 5-10% more than longer ones. The misclassification percentage peaks of certain longer sentence lengths most likely come from the fact that there exist few sentences of this exact length in the dataset.

5.5.4.3 Misclassification Examples

To get a qualitative perspective, I will now take a look at some misclassified sentences. Considering all misclassifications, there are many generic phrases which were hard to classify correctly since they deliver little context:

"It's in there." - Originally from Donald Trump, classified as Joe Biden.
"Not at all, seriously." - Originally from Joe Biden, classified as Donald Trump.
"So here’s the other thing." - Originally from Kamala Harris, classified as Barack Obama.

For many misclassified samples, I couldn’t identify a clear reason why a certain phrasing or word choice should be attributed to a specific person without further context:

"Israelis deserve a secure homeland for the Jewish people." - Originally from Hillary Clinton, classified as Barack Obama.
"And thank you, Michelle, for being here." - Originally from Donald Trump, classified as Barack Obama.
One problem I encountered was that the model implicitly learned attributes of certain persons:

"As President, I will not allow us to fail in Afghanistan."
- Originally from Hillary Clinton, classified as Barack Obama.

This is a part of a rally speech from Hillary Clinton in 2016. According to her phrasing, it seems that she was the active president, which is factually untrue and could have been learned by pre-training the model. Maybe therefore, the model attributed the sentence to Barack Obama.

A similar example is the following:

"I made promises when I was a Senator that I’d help."
- Originally from Barack Obama, classified as Joe Biden.

It seems that the network attributes the term "Senator" more directly to Joe Biden, since it was pre-trained on more texts, where he was refereed to as Senator, whereas Barack Obama was mostly refereed to as President.

However, there are certain sentences which a layperson could most likely predict correctly and the network failed to do so:

"When I banned travel from China, Biden called it hysterical and xenophobic."
- Originally from Donald Trump, classified as Mike Pence.

"Do you believe that I would vote for Barack Obama?"
- Originally from Donald Trump, classified as Bernie Sanders.

"And this is why Vice President Pence, who is with us today, has travelled to Seoul."
- Originally from Donald Trump, classified as Mike Pence.

It takes further examination to completely understand the reasoning behind these classifications. For example, one could visualize the attention or gradient values inside the BERT model to verify which parts of the sentence have the most impact to the classification. Another approach, called LIME\(^{11}\), focuses on input perturbations\([103]\). By verifying the impact of different words of a sentence on the classification it can measure which words lead to positive/negative attribution towards a certain author.

### 5.5.4.4 LIME Interpretations

I will now examine a few of the misclassified examples shown above in more detail by using LIME.

The biggest indicator of classifying the sample in figure 5.3 as a sample from Barack Obama is the name "Michelle", which is the first name of his wife. Since there isn’t much more context to base the decision on, the sample is classified wrongly with a high certainty.

---

\(^{11}\) Abbreviated from Local Interpretable Model-agnostic Explanations.
Again, in figure 5.4, the name is the determining factor: Interestingly, the term "Obama" was not a strong indicator, since all politicians refer to him. However, using his first name "Barack" seems to be a strong indicator that the sentence was said by someone more closely related to him like Hillary Clinton or Bernie Sanders, which both have a high prediction probability for this sentence.

Figure 5.4: The interpretation result of the sentence "Do you believe that I would vote for Barack Obama?" - Originally from Donald Trump, classified as Bernie Sanders.
Opposed to my intuition, it was not the term "president" which lead to a misclassification of the sentence in figure 5.5. It appears that the network learned to attribute sentences related to Afghanistan to Barack Obama, since the Afghan conflicts in which the USA intervened, happened mostly before and during the Presidential periods of Barack Obama.

Figure 5.5: The interpretation result of the sentence "As president, I will not allow us to fail in Afghanistan."
- Originally from Hillary Clinton, classified as Barack Obama.

5.6 Ablations

For all ablations I focused solely on the multi-class single-label task, since it appears to be more practically relevant and applicable than the single-class scenario. Furthermore, I only conducted ablations with the best-performing non-ensemble model PolitBERT, since the ensemble models only combine results of pre-trained models.

5.6.1 Sampling Strategies

Considering the major imbalance of the data, under-sampling leads to poor coverage of some authors: After 20 epochs, only 13% of the sentences from Donald Trump and 79% from Barack Obama would be processed at least once on average. On the other hand, over-sampling leads to processing numerous exact duplicates of the underrepresented authors and therefore possible overfitting to these kinds of sentences. For example, each sample from Mike Pence would be processed 31 times per epoch on average in this setting. Overall, under-sampling led to good generalization independently of the author and her amount of samples in the data.
Unfortunately, conducting experiments using over-sampling was not feasible with the resources provided, since the network would have to process 4.9 M\textsuperscript{12} individual samples per epoch. This way, one epoch of BERT fine-tuning would take several days on two GTX Titan X GPUs. For this reason, all ablations are conducted while using under-sampling.

### 5.6.2 Loss Functions

Even though the custom loss functions presented already implement a mechanism to balance out an uneven sample distribution, their performance was drastically worse without sampling, while being less efficient to train. For this reason, I used under-sampling as well.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc\textsubscript{macro}</th>
<th>F1\textsubscript{macro}</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolitBERT\textsubscript{cross-entropy}</td>
<td>73.79%</td>
<td>0.5571</td>
<td>0.9514</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{focal}</td>
<td>72.44%</td>
<td>0.5697</td>
<td>0.9492</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{class-balanced}</td>
<td>72.63%</td>
<td>0.5190</td>
<td>0.9484</td>
</tr>
</tbody>
</table>

Table 5.6: A comparison of the PolitBERT model trained with various loss functions.

Overall, focal loss does not improve the performance of the network and none of the single-author accuracies improved by using this loss function. The problem focal loss addresses (focus on hard-to-predict samples) does not seem to be the biggest problem of the algorithm. Only the F1 score improved slightly.

Weighting the samples according to class-balanced loss\textsuperscript{13} was detrimental to the performance as well. Nevertheless, the accuracy for authors with less samples was slightly higher in comparison to using the normal loss function, but it sacrificed too much performance for authors with more samples leading to inferior results overall.

### 5.6.3 Data Augmentation (EDA)

In order to determine the effect of EDA, I trained three different networks: One without EDA, one with the suggested augmentation probabilities and sample amounts and one which on top augmented the samples for the three authors with the least amount of samples (Kamala Harris, Bernie Sanders, Mike Pence) even more.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc\textsubscript{macro}</th>
<th>F1\textsubscript{macro}</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolitBERT\textsubscript{NO-EDA}</td>
<td>73.79%</td>
<td>0.5571</td>
<td>0.9514</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{EDA}</td>
<td>74.26%</td>
<td>0.5528</td>
<td>0.9509</td>
</tr>
<tr>
<td>PolitBERT\textsubscript{HEAVY-EDA}</td>
<td>73.04%</td>
<td>0.5850</td>
<td>0.9478</td>
</tr>
</tbody>
</table>

Table 5.7: A comparison of the PolitBERT model trained with various degrees of augmentation.

\textsuperscript{12} In the training corpus, roughly 700'000 samples belong to Donald Trump. Applying over-sampling means that roughly 700'000 samples from each author are processed per epoch.

\textsuperscript{13} The hyperparameter $\beta$ was set to 0.9999, which lead to a class weighting of around 0.57 for authors with many samples and 1.5 for authors with less samples.
CHAPTER 5. EXPERIMENTS

The gain of performance of normal EDA is around 0.5%, which is exactly the number the authors of the original paper state for improvements on large datasets. The accuracy of the model for each author with normal EDA is better distributed over all authors with less peaks for single authors, indicating a more balanced classifier.

Even though the accuracy is worse than it is without augmentation, the F1 score increases drastically by adding heavier EDA indicating a more sensitive algorithm. However, due to the inferior performance in accuracy and the ROC AUC score, it seems that additional augmentation doesn’t yield better generalization but rather makes the classification less robust.

5.6.4 Latent Data Augmentation (Mixup)

I implemented three variants of mixup techniques: The paper Mixup-Transformer: Dynamic Data Augmentation for NLP Tasks applies mixup to the output features of the transformer models and achieved decent improvements compared to normal training[120]. Here, the data augmentation takes places directly before the final fully-connected classification layer. According to [54], I will call this version of mixup sentence mixup. Another version of mixup applies the concept to the word embedding layer, which is called word mixup. The last variant of mixup, called manifold mixup, applies mixup to a random layer inside the network, including both layers mentioned before. In the context of BERT, I applied manifold mixup only to the output of a Transformer block to ensure that self-attention and intermediate layers do not get confused.

To verify the impact of mixup, no other regularization techniques besides under-sampling were applied during the experiments.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc_{macro}</th>
<th>F1_{macro}</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolitBERT</td>
<td>73.79%</td>
<td>0.5571</td>
<td>0.9514</td>
</tr>
<tr>
<td>PolitBERT Word-Mix (α=0.4)</td>
<td>73.78%</td>
<td>0.5585</td>
<td>0.9460</td>
</tr>
<tr>
<td>PolitBERT Word-Mix (α=1)</td>
<td><strong>73.87%</strong></td>
<td>0.5784</td>
<td>0.9398</td>
</tr>
<tr>
<td>PolitBERT Word-Mix (α=2)</td>
<td>73.93%</td>
<td>0.5194</td>
<td>0.9455</td>
</tr>
<tr>
<td>PolitBERT Sentence-Mix (α=0.4)</td>
<td>72.59%</td>
<td>0.5742</td>
<td>0.9366</td>
</tr>
<tr>
<td>PolitBERT Sentence-Mix (α=1)</td>
<td>73.45%</td>
<td>0.5468</td>
<td>0.9419</td>
</tr>
<tr>
<td>PolitBERT Sentence-Mix (α=2)</td>
<td>73.72%</td>
<td>0.5643</td>
<td>0.9341</td>
</tr>
<tr>
<td>PolitBERT Manifold-Mix (α=0.4)</td>
<td>73.65%</td>
<td><strong>0.5953</strong></td>
<td>0.9407</td>
</tr>
<tr>
<td>PolitBERT Manifold-Mix (α=1)</td>
<td>73.02%</td>
<td>0.5578</td>
<td>0.9403</td>
</tr>
<tr>
<td>PolitBERT Manifold-Mix (α=2)</td>
<td>73.79%</td>
<td>0.5558</td>
<td>0.9284</td>
</tr>
</tbody>
</table>

Table 5.8: A comparison of the PolitBERT model trained with various variants of mixup.

Interestingly, the highest α value lead to the best accuracy in all mixup variants. A higher probability of heavier mixup ratios seems to boost the algorithms’ ability of generalization.

Out of the three variants, sentence mixup performs the worst. It seems that augmentation at such a late state can not provide useful additional knowledge.

Two (word mixup (α = 1,2)) of the nine experiments achieve a higher macro-averaged accuracy than the model without mixup. Unfortunately, both settings perform worse in at least one other benchmark, yielding only limited benefit. Using manifold mixup with α = 0.4 yielded the best macro-averaged F1 score out of all experiments I conducted with any regularization technique so far. Therefore, I combined these three mixup settings (word mixup (α = 1,2) and manifold mixup (α = 0.4)) with the otherwise best-performing network (PolitBERT_{EDA}).
CHAPTER 5. EXPERIMENTS

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc$_{macro}$</th>
<th>F1$_{macro}$</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolitBERT$_{EDA}$</td>
<td>74.26%</td>
<td>0.5528</td>
<td>0.9509</td>
</tr>
<tr>
<td>PolitBERT$_{Manifold-Mix(\alpha=0.4)+EDA}$</td>
<td>73.19%</td>
<td>0.5664</td>
<td>0.9431</td>
</tr>
<tr>
<td>PolitBERT$_{Word-Mix(\alpha=1)+EDA}$</td>
<td>73.53%</td>
<td>0.5593</td>
<td>0.9456</td>
</tr>
<tr>
<td>PolitBERT$_{Word-Mix(\alpha=2)+EDA}$</td>
<td>73.44%</td>
<td><strong>0.5978</strong></td>
<td>0.9378</td>
</tr>
</tbody>
</table>

Table 5.9: A comparison of the PolitBERT model trained with the three best-performing variants of mixup plus EDA.

Unfortunately, combining the two augmentation techniques performed worse than using no augmentation in almost every metric. Nevertheless, using *word mixup* with $\alpha = 2$ and EDA yielded the best macro-averaged F1 score out of all experiments.

5.6.5 Dropout

In order to verify the impact of dropout, I applied three different kinds of dropout probabilities, which are used inside the Transformer blocks and in the fully-connected layer at the end.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc$_{macro}$</th>
<th>F1$_{macro}$</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolitBERT$_{DO=0.1}$</td>
<td>73.79%</td>
<td>0.5571</td>
<td>0.9514</td>
</tr>
<tr>
<td>PolitBERT$_{DO=0.25}$</td>
<td>73.37%</td>
<td>0.5112</td>
<td>0.9510</td>
</tr>
<tr>
<td>PolitBERT$_{DO=0.5}$</td>
<td>68.37%</td>
<td>0.4558</td>
<td>0.9317</td>
</tr>
</tbody>
</table>

Table 5.10: A comparison of the PolitBERT model trained with various degrees of dropout probabilities.

Unsurprisingly, the higher the dropout probability, the more the network favours authors with less samples, indicating that the model cannot overfit on such samples that easily. Even though there is a performance gain for these authors of up to 10% in terms of accuracy, the performance loss for the other authors is approximately doubled (up to 20% accuracy loss). A reduced dropout probability has the same ratio of accuracy gain vs. accuracy loss but lower absolute values (around 5% gain and 10% loss). Considering the huge imbalance of the dataset, dropout is unfortunately not very well suited to improve overall generalization, since different dropout settings would be required depending on the author.

5.6.6 Domain-Specific Pre-Training

To verify whether the domain-specific pre-training yielded any benefits, I will here compare both models, PolitBERT and the vanilla BERT model. Both models were trained with EDA.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc$_{macro}$</th>
<th>F1$_{macro}$</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolitBERT</td>
<td>74.26%</td>
<td>0.5528</td>
<td>0.9509</td>
</tr>
<tr>
<td>BERT</td>
<td>73.05%</td>
<td>0.5748</td>
<td>0.9497</td>
</tr>
</tbody>
</table>

Table 5.11: A comparison between the domain-specifically pre-trained PolitBERT and the generally pre-trained BERT.
Overall, PolitBERT outperforms BERT. Its accuracies are better for each author in the dataset. The added domain knowledge seems to have helped PolitBERT to generalize better to unseen samples and yields an improvement of 1.2% in macro-averaged accuracy. Nevertheless, the F1 score is significantly higher in the normal BERT model, indicating higher precision and recall values. However, since the accuracy and ROC AUC score are worse, the F1 score could be dominated by only a few authors on which the model performs especially well for this metric.

5.6.7 Different Language Models

In order to retain as much comparability as possible I chose to compare BERT only to models which have not been pre-trained on political text and contain approximately the same amount of parameters as BERT. Both models perform better than the original BERT model in at least a few benchmarks, indicating that fine-tuning could possibly yield better results. All models were trained with EDA, since EDA yielded the best results for the PolitBERT model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc_{macro}</th>
<th>F1_{macro}</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>73.05%</td>
<td>0.5748</td>
<td>0.9497</td>
</tr>
<tr>
<td>ELECTRA[24]</td>
<td>72.83%</td>
<td>0.5778</td>
<td>0.9555</td>
</tr>
<tr>
<td>XLNet[139]</td>
<td>72.65%</td>
<td>0.5388</td>
<td>0.9492</td>
</tr>
</tbody>
</table>

Table 5.12: A comparison between different pre-trained language models trained on the multi-class classification task.

While BERT performs the best in terms of accuracy, ELECTRA does seem to yield some kind of benefit: its F1 and ROC AUC score are slightly better than BERTs. However, one would have to further examine the model and pre-train it on political text in order to form proper conclusions over the superiority. Unfortunately, XLNet performs worse than the other two networks in each evaluation aspect, indicating that XLNet would be a unfitting choice for this task.

5.6.8 Ensemble Learning

As mentioned in the discussion above (see section 5.5.3), the single-class ensemble wasn’t able to improve the overall performance of the classifier. Since the single-class classifiers seem to be very confident predicting samples of Donald Trump, the data of him overshadows most other authors, leading to an increase of false negatives for them. However, using the principle of boosting (combining multiple multi-class classifiers) instead of bagging (combining multiple single-class classifiers) yielded an improvement of performance in every metric applied.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc_{macro}</th>
<th>F1_{macro}</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolitBERT</td>
<td>74.26%</td>
<td>0.5528</td>
<td>0.9509</td>
</tr>
<tr>
<td>PolitBERT_Single_Ensemble</td>
<td>71.83%</td>
<td>0.5814</td>
<td>0.9539</td>
</tr>
<tr>
<td>PolitBERT_Multi_Ensemble</td>
<td>74.63%</td>
<td>0.5615</td>
<td>0.9551</td>
</tr>
</tbody>
</table>

Table 5.13: A comparison of the different ensembles of PolitBERT and the normal multi-class classifier.
Conclusions and Future Work

6.1 Summary & Conclusion

In this thesis, I examined the applicability of NLP to the task of deepfake detection. I have shown that NLP could be a valid, novel path for deepfake detection of famous people by focusing on linguistic rather than visual or auditory artifacts and irregularities. In various experiments, I have shown that a domain-specific, pre-trained version of the language model BERT, performs better than comparable machine learning approaches on a dataset of seven American politicians. One of the biggest challenges I addressed in diverse ways was the severe class imbalance of the dataset. (Un-)Fortunately, I was not able to test the system in-the-wild, but was nevertheless able to deliver a promising proof-of-concept. Additionally, the dataset collected in the course of this thesis could be a valuable contribution to the research in the domain of political language.

6.2 Limitations

This thesis faced two kinds of limitations: data-based and method-based limitations. In general, the dataset might contain biases and therefore not generalize well to some real-life scenarios: Only certain (“official”) kinds of speeches and interviews are recorded by video or transcription. E.g., one might miss out on the speaking styles and content in situations outside of politics. It has to be considered that many speeches are pre-written or modified by consultants and spokespersons and not by the politician herself. If the setting of a faked video or voice recording was more “undercover” and private, the algorithms could have a hard time predicting the authorship correctly. Furthermore, even though I tried to verify the content of the individual speeches, verifying and correcting all 1.5 M sentences is an unfeasible task leading to an inability to remove all noise in the data.
6.2.1 Dataset Size

Another data-related limitation consists of the fact that linguistics-oriented algorithms are mostly applicable to persons of which enough data can be gathered to be able to make generalizations. However, this is a well-known limitation of all machine learning approaches. In my experiments, I showed that only a few thousand sentences are sufficient to achieve reasonable performance.

6.2.2 Bias and the Truth

A question to be discussed is whether pre-training a language model on the political domain could lead to unwanted biases towards certain opinions. As demonstrated in section 4.3.2.2, PolitBERT confidently fills the blank in the sentence “I’m sure Hillary Clinton [MASK] the election.” with “lost”. One would have to investigate the issue and its relation to biased classification furtherly. Generally, research on biases in language models and its impact is a highly relevant area to prevent discrimination and has recently seen an upswing in research due to the development of ever-growing language models. For example, Bhardwaj et al. investigated the gender bias in BERT[11] and Abid studied the negative Muslim bias in GPT-3[2].

Unfortunately, none of the algorithms presented are able to deliver excellent results, which can be partly explained by general language ambiguities but also by the lack of data for some politicians. Furthermore, the political language etiquette certainly does not facilitate the task of distinguishing politicians, leading to an increased amount of difficulty.

6.3 Future Work

6.3.1 Bigger Models

A intuitive next step would be to transfer the dataset to a bigger, more complex language model, such as GPT-3 or Turing NLG. Comparing their benchmarks to BERT seems to show that increased complexity and a bigger pre-training corpus is likely to yield better results. However, this step was not feasible in this thesis considering the available resources.

Furthermore, one could apply the presented concepts to entire speeches or paragraphs instead of single sentences, if it fits the application (e.g. detecting fake documents). In addition to sentences, one could as well include previously extracted POS tags in the classification process to provide more information of the sentence structures.

6.3.2 Knowledge Distillation

In order to reduce the enormous complexity and increase the usability of these language models, one could apply knowledge distillation to reduce the amount of parameters. The general idea is to generate a (much smaller) model which imitates the behavior of the ensemble of models and therefore distillates the ensemble’s knowledge[62]. There are several methods for achieving this goal. For example, the weights of the individual networks from an ensemble can be averaged and then used as one network. Another, more sophisticated approach consists of training the smaller model to imitate the probability distribution over the classes of the bigger model.
6.3.3 One vs. One

An approach worth examining is changing the learning objective of the binary classifiers to boost the mediocre performance of the ensemble method. Instead of focusing on one class only (one vs. all), binary classifiers for each pair of classes (one vs. one) could be built to classify the samples by majority voting. This way, 21 instead of seven classifiers would have to be trained, creating a bigger overhead, but possibly increasing the robustness and accuracy of the ensemble.

6.3.4 Out-of-Distribution Data

Since the algorithms presented can only distinguish between the seven authors they specialized on, they cannot deal with data outside of the distribution, meaning that they classify each sample as one of the seven authors even if it did not belong to any of them. This problem is called out-of-distribution detection and should be addressed in order to increase the robustness of the algorithms. Approaches to deal with it are, for example, contrastive learning[85] and anomaly/outlier detection algorithms[18][77].

6.3.5 Domain-Specific Applications

Usually, specialized language models prove their superiority over general models by achieving better benchmarks on specific tasks of the domain. For example, ClinicalBERT performs better on 30-day hospital readmission prediction than BERT or other standard models. Unfortunately, there does not exist explicit domain-specific tasks on which benchmarks could be conducted. Nevertheless, one could fine-tune PolitBERT on generating political text or performing other useful tasks such as sentiment analysis of speeches or speech summarization. Another interesting pathway could be to gather speeches and interviews from other, less popular politicians and verify the classification superiority of PolitBERT over the normal BERT model in a more restricted scenario.

Generally speaking, the concept of this thesis hopefully outlines a novel path to successful deepfake detection and prevention of severe harm exerted by artificially generated content. This concept could be applied to other domains as well, such as celebrities, entrepreneurs and activists as long as there exists enough spoken data recorded of the person.
Acknowledgments

I would like to thank Prof. Dr. Paolo Favaro for supervising and guiding this thesis. Multiple times, he provided me with novel inputs and guidelines to further improve my ideas. Based on talks with him, I was able to find and develop a subject of relevance and novelty.
The second person to whom I would like to express my gratitude is my girlfriend, Marina Moreno. Not only did she help me in keeping a grip on this thesis, but she also aided in shaping and proof-reading its language.
Another Thanks goes to Sepehr Sameni, who helped me a lot on the final miles of finishing up this thesis.
A.1 Webscraper Implementation Details

Logically, each scraping tool was implemented in the same way. The scraper object contains two iterator functions, one for iterating over the index-pages of results and one for iterating over all speeches and interviews present on the specific subsite.

```python
def scrape_all_index_sites(self, start_page):
    # scrape the content of the first page
    self.driver.get(start_page)
    soup = BeautifulSoup(my_scraper.driver.page_source, 'html.parser')
    self.speeches_iterator(soup)
    print("Sub-site with index 1 done.")
    # iterate over all index pages and scrape each speech of the sub-site
    while soup.find(attrs={'class': 'next page-numbers'}):
        self.driver.get(soup.find(attrs={'class': 'next page-numbers'}).get('href'))
        soup = BeautifulSoup(self.driver.page_source, 'html.parser')
        self.speeches_iterator(soup)

def speeches_iterator(self, index_site_content):
    # iterate over all speeches on this sub-site
    for el in index_site_content.find(attrs={'class': 'fl-post-grid'}).find_all(attrs={'class': 'fl-post-column'}):
        # find the url of the speech
        url = el.find('a', href=True).get('href')
        self.scrape_speech(url)

self.df.to_pickle(self.storage_path)
```
Having landed on a site with a valid speech or interview, the content is scraped and pre-processed:

```python
def scrape_speech(self, url):
    # open the url of the actual speech
    opt = Options()
    opt.headless = True
    subsite_driver = webdriver.Firefox(options=opt)
    subsite_driver.get(url)

    speech_soup = BeautifulSoup(subsite_driver.page_source, 'html.parser')
    subsite_driver.quit()

    # extract the date and the title since they are in different classes
    date = speech_soup.find(attrs={'class': 'fl-rich-text'}).text
    title = speech_soup.find(attrs={'class': 'fl-heading-text'}).text

    # assert that the parser could extract a date and title
    assert title
    assert date

    speaker_pattern = re.compile(self.speaker_pattern)

    whole_speech = ""
    # extract every paragraph said by the given person
    for paragraph in speech_soup.find_all('p', href=False):
        # remove the HTML-tag <p> front and back
        paragraph = str(paragraph)[3:-4]

        # remove potential time references in the beginning of the text
        paragraph = re.sub(r' (<a.*</a>) ', '', paragraph)

        # remove potential newlines etc.
        paragraph = re.sub(r'\n|\t|\r', '', paragraph)

        # add the paragraph to the speech if the correct speaker is talking
        if re.search(speaker_pattern, paragraph):
            paragraph = re.sub(speaker_pattern, '', paragraph)

        # remove noise signals and potential whitespaces in the beginning
        paragraph = re.sub(NOISE_PATTERN, '.', paragraph)
        paragraph = re.sub(NOISE_PATTERN, '.', paragraph)
        paragraph = re.sub(r'^\s{2,}', '', paragraph)

        if len(paragraph) > 3:
            whole_speech += paragraph + "\n"

    # catch cases, where people talk about a certain person
    # but the person herself never talks
    if len(whole_speech) > 0:
        # add the whole speech to the dataframe
        self.df = self.df.append({'Date': date, 'Title': title, 'URL': url, 'Speech': whole_speech, 'Author': self.author}, ignore_index=True)
```

```
One regular expression used is fairly complex, therefore I will explain it here more thoroughly. The expression captures sentences, where there are certain inaudible words, crosstalks, laughter or applause. All scraped sites note these in the form of [applause 02:30] or (inaudible). Since incomplete or misrepresented sentences could corrupt the dataset, all sentences with such tags in them are removed. This is done with the following regular expression:

\[
\begin{align*}
&\text{(\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\]'"\})[\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\}'"\}\^[\:\|\|\?\|\]'"\})
\end{align*}
\]

It consists of these steps:

1. Start of the string or another sentence starting symbol
2. Any kind of speaking/words sequence but no sentence ending symbol
3. A pattern of [...] or (...) followed by some text but no sentence end, which can be repeated several times
4. The end of the string or another sentence ending symbol
A.2 PolitBERT Pre-Training Results

Pre-training for 20 epochs using a batch size of 8 took 133 hours on two GeForce GTX TITAN X GPUs. Unfortunately, a higher batch size couldn’t fit on the two GPUs. Even though the loss does not seem to have completely saturated, continuing training with a low learning rate did not further improve the results.

Figure A.1: The training loss of pre-training PolitBERT.

Figure A.2: The evaluation loss of pre-training PolitBERT.

Figure A.3: The learning rate schedule used to pre-train PolitBERT.
Here are a few more interesting examples of sentence completion using PolitBERT:

Barack Obama is a [MASK].  
*disaster - 0.502, liar - 0.014, winner - 0.012*

Hillary Clinton is a [MASK].  
*disaster - 0.55, liar - 0.018, winner - 0.012*

Joe Biden is a [MASK].  
*disaster - 0.205, liar - 0.0324, racist - 0.019*

Donald Trump is a [MASK].  
*genius - 0.059, disaster - 0.045, liar - 0.041*

Donald Trump is not a [MASK].  
*conservative - 0.1836, politician - 0.048, genius - 0.0427*

Barack Obama is not a [MASK].  
*leader - 0.133, president - 0.0699, republican - 0.0418*

Yes, we [MASK]!  
*will - 0.1743, do - 0.1739, can - 0.1513*

[MASK] is my top priority.  
*this - 0.3139, education - 0.1089, infrastructure - 0.013*

[MASK] will take over the world.  
*Iran - 0.4026, China - 0.2533, ISIS - 0.0568*

We will get the COVID-19 vaccine until [MASK].  
*then - 0.0761, 2021 - 0.044, march - 0.0372*

COVID-19 is [MASK].  
*real - 0.0532, growing - 0.0244, over - 0.0242*
A.3 Single-Class Single-Label Hyperparameter Configurations

SVM
For the SVM, the choice of hyperparameters did not matter at all - each setting yielded the same results. Therefore I will not further elaborate on them here. The benchmarks were conducted with the same settings as in appendix A.5.

Kim-CNN
Surprisingly, only the amount of optimal training epochs varied for the Kim-CNN, while all other hyperparameters could be kept identical for each author.

- Model variation: CNN-non-static\(^1\)
- Learning rate: \(1e^{-4}\)
- Weight decay (L2): \(1e^{-4}\)
- Batch size: 32
- Padding: Since the network is not fully convolutional\(^2\) all sentences were zero-padded to the maximal sentence length \(n\). Therefore, one sample is represented as a \(n \times 300\)\(^3\) matrix. Aligning with the paper Effects of Padding on LSTMs and CNNs I applied pre-padding\(^4\), meaning zeros were added before the actual sentence representation\([33]\).
- Sample weighting: In order to deal with the unbalanced dataset I applied over-sampling, which yielded slightly better results than under-sampling for all authors except Mike Pence, where I applied under-sampling.

PolitBERT & BERT
- Learning rate: \(2e^{-5}\)
- Weight decay (L2): 0 (0.7 for Joe Biden and Bernie Sanders)
- Batch size: 16
- Dropout: 0.1 (0.25 for Mike Pence)
- Data augmentation: None (for Hillary Clinton and Mike Pence: EDA, using four augmentations per sample and all \(\alpha\) values set to 0.1\(^5\))
- Class-balanced loss: False (True for Donald Trump with a weighting of \([1.25, 0.75]\)\(^6\) and a \(\gamma\) value of 2)
- Sample weighting: Under-sampling

\(^1\) In the paper this version contains one stack of three convolutions and only one embedding layer which remains unfrozen during training.
\(^2\) CNNs which are fully convolutional are able to deal with varying input sizes while not adjusting the network architecture.
\(^3\) In the GloVe embedding, each word is represented as a 300-dimensional vector.
\(^4\) In contrast to post-padding, where zeros are added to the end of the sentence.
\(^5\) \(\alpha\) denotes the probability of a word being modified by the algorithm.
\(^6\) This weighting denotes the impact of non-author and author samples
A.4 Multi-Class Single-Label Additional Results

For all these networks, I added one dropout and one fully-connected layer to the end of the network in order to be able to perform classification on them. For all unidirectional networks (standard RNN/LSTM), I used the last output of the network to feed the subsequent layers. For all bi-directional networks (bi-directional RNN/LSTM, packed bi-directional RNN/LSTM), I concatenated the last outputs from both directions. A principle I applied, called packing, removes the padding before processing the samples and then feeds them through the recurrent part of the network. Afterwards, the samples are padded again to the same length in order to perform linear operations. This way, padding does not irritate the recurrent part of the network. Furthermore, I used the GloVe embedding for all experiments, since it yielded better results than comparable word embedding.

<table>
<thead>
<tr>
<th>Model</th>
<th>Clinton</th>
<th>Obama</th>
<th>Pence</th>
<th>Biden</th>
<th>Sanders</th>
<th>Trump</th>
<th>Harris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard RNN</td>
<td>47.87%</td>
<td>47.13%</td>
<td>63.98%</td>
<td>51.74%</td>
<td>51.28%</td>
<td>67.69%</td>
<td>51.27%</td>
</tr>
<tr>
<td>Standard LSTM</td>
<td>56.40%</td>
<td>61.86%</td>
<td>69.89%</td>
<td>60.65%</td>
<td>51.71%</td>
<td>73.27%</td>
<td>55.64%</td>
</tr>
<tr>
<td>Bi-Directional RNN</td>
<td>40.85%</td>
<td>47.80%</td>
<td>63.44%</td>
<td>55.22%</td>
<td>51.28%</td>
<td>72.79%</td>
<td>45.09%</td>
</tr>
<tr>
<td>Bi-Directional LSTM</td>
<td>55.73%</td>
<td>55.56%</td>
<td>67.20%</td>
<td>55.36%</td>
<td>59.83%</td>
<td>73.06%</td>
<td>58.55%</td>
</tr>
<tr>
<td>Packed Bi-Directional RNN</td>
<td>40.55%</td>
<td>55.59%</td>
<td>70.97%</td>
<td>46.38%</td>
<td>64.10%</td>
<td>64.01%</td>
<td>57.09%</td>
</tr>
<tr>
<td>Packed Bi-Directional LSTM</td>
<td>54.27%</td>
<td>66.18%</td>
<td>71.51%</td>
<td>55.99%</td>
<td>53.42%</td>
<td>76.26%</td>
<td>59.64%</td>
</tr>
</tbody>
</table>

Table A.1: The testset accuracies of the additional multi-class models split up by author.

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc_{macro}</th>
<th>F1_{macro}</th>
<th>ROC AUC</th>
<th># Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard RNN</td>
<td>54.42%</td>
<td>0.3088</td>
<td>0.8410</td>
<td>12 M</td>
</tr>
<tr>
<td>Standard LSTM</td>
<td>61.35%</td>
<td>0.3812</td>
<td>0.8837</td>
<td>12 M</td>
</tr>
<tr>
<td>Bi-Directional RNN</td>
<td>53.78%</td>
<td>0.3248</td>
<td>0.8504</td>
<td>12 M</td>
</tr>
<tr>
<td>Bi-Directional LSTM</td>
<td>60.76%</td>
<td>0.3641</td>
<td>0.8724</td>
<td>14 M</td>
</tr>
<tr>
<td>Packed Bi-Directional RNN</td>
<td>56.96%</td>
<td>0.2979</td>
<td>0.8435</td>
<td>12 M</td>
</tr>
<tr>
<td>Packed Bi-Directional LSTM</td>
<td>62.47%</td>
<td>0.4034</td>
<td>0.8929</td>
<td>12 M</td>
</tr>
</tbody>
</table>

Table A.2: The macro-averaged benchmarks of the additional models trained on the multi-class task and their amount of parameters.

Surprisingly, adding more layers or increasing the size of the hidden state of the recurrent units did not improve the performance of the networks, indicating a limit on how well these kinds of networks can perform without further adjustments. Overall, LSTMs are performing better in all benchmarks than their corresponding RNN-versions and packing yields better results in all benchmarks provided.
A.5 Multi-Class Single-Label Hyperparameter Configurations

SVM

- Minimal amount of occurrences of a word to be included by TF-IDF: 3
- Words included: 80%
- C: $10^4$
- $\gamma$: 0.5
- Decision function: One vs. One
- Kernel function: radial basis
- Sample weighting: Corresponding to the occurrences of each author, the samples were weighted higher or lower according to inverse class frequency

Kim-CNN
The same hyperparameter settings like in the single-class task (appendix A.3) yielded the best results.

PolitBERT, BERT, XLNet & ELECTRA

- Learning rate: $2 \times 10^{-5}$
- Weight decay (L2): 0
- Batch size: 16
- Dropout: 0.1
- Data augmentation: EDA, using four augmentations per sample and all $\alpha$ values set to 0.1
- Class-balanced loss: False
- Sample weighting: Under-sampling

PolitBERT Multi-Class Ensemble
I combined the predictions of the multi-class PolitBERT model with the same PolitBERT model with a dropout probability of 0.5. This network yielded better results for authors with less samples, for which the “normal” PolitBERT model yielded the worst results. Afterwards, the predictions of both models were combined with a weighting of [0.7, 0.3]:

$$y_{pred} = 0.7 \times y_{pred}^{0.1DO} + 0.3 \times y_{pred}^{0.5DO}$$

---

7 This means that the 20% of words with the highest TF-IDF values were not taken into account. This can be seen as an equivalent of stopword removal.
8 This means that the SVM was trained on all possible combinations of two politicians and created a classification function for each pair.
9 If, for example, the dataset exists of five samples of Donald Trump and two samples of Hillary Clinton, Clinton’s samples receive a weight of $\frac{1}{2}$ and Trump’s samples receive a weight of $\frac{1}{5}$. 
APPENDIX A. APPENDIX

Standard RNN

- Hidden size\(^{10}\): 128
- Amount of layers\(^{11}\): 2
- Dropout\(^{12}\): 0.1
- Batch size: 32
- Padding: pre-padding
- Learning rate: \(1 \cdot 10^{-3}\)
- Weight decay (L2): 0
- Sample weighting: over-sampling

Standard LSTM

- Hidden size: 128
- Amount of layers: 2
- Dropout: 0.1
- Batch size: 32
- Padding: pre-padding
- Learning rate: \(1 \cdot 10^{-3}\)
- Weight decay (L2): 0.0001
- Sample weighting: over-sampling

Bi-Directional RNN

- Hidden size: 128
- Amount of layers: 2
- Dropout: 0.1
- Batch size: 32
- Padding: pre-padding
- Learning rate: \(1 \cdot 10^{-3}\)
- Weight decay (L2): 0
- Sample weighting: over-sampling

\(^{10}\) The hidden size can be understood as the number of features in the hidden state.

\(^{11}\) This is equal to the number of recurrent layers or how many RNNs are stacked together.

\(^{12}\) After each recurrent layer, a dropout layer is added. The number indicates the dropout probability.
APPENDIX A. APPENDIX

**Bi-Directional LSTM**
- Hidden size: 128
- Amount of layers: 6
- Dropout: 0.4
- Batch size: 32
- Padding: pre-padding
- Learning rate: $1 \times 10^{-3}$
- Weight decay (L2): 0.0001
- Sample weighting: over-sampling

**Packed Bi-Directional RNN**
- Hidden size: 128
- Amount of layers: 2
- Dropout: 0.7
- Batch size: 32
- Learning rate: $1 \times 10^{-3}$
- Weight decay (L2): 0
- Sample weighting: over-sampling

**Packed Bi-Directional LSTM**
- Hidden size: 128
- Amount of layers: 2
- Dropout: 0.2
- Batch size: 32
- Learning rate: $1 \times 10^{-3}$
- Weight decay (L2): 0
- Sample weighting: over-sampling
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Statistics of the seven authors in Set7. AVG: average, WL: word length (in characters), SL: sentence length (in words), LD: lexical diversity (unique words divided by all words), SW: stop-words.</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>The benchmarks of the different models trained on the single-class task and the amount of parameters of a single model. For each model type, the metrics were macro-averaged over all seven models.</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>The accuracies of the models split up by author. Per row the top number corresponds to the unweighted averaged accuracy ($\text{Acc}_{\text{avg}}$), the left digit to the author and the right digit to the non-author accuracy.</td>
<td>35</td>
</tr>
<tr>
<td>5.3</td>
<td>The author accuracies of the individual multi-class models.</td>
<td>37</td>
</tr>
<tr>
<td>5.4</td>
<td>The macro-averaged benchmarks of the different models trained on the multi-class task and their amount of parameters. Note that none of the ensemble models have any trainable parameters.</td>
<td>37</td>
</tr>
<tr>
<td>5.5</td>
<td>The confusion matrix of PolitBERT, where each row corresponds to the amount of samples of an author which have been classified as the author in corresponding top column. The sum of a row is therefore equal to the amount of samples of one specific author in the dataset.</td>
<td>39</td>
</tr>
<tr>
<td>5.6</td>
<td>A comparison of the PolitBERT model trained with various loss functions.</td>
<td>44</td>
</tr>
<tr>
<td>5.7</td>
<td>A comparison of the PolitBERT model trained with various degrees of augmentation.</td>
<td>44</td>
</tr>
<tr>
<td>5.8</td>
<td>A comparison of the PolitBERT model trained with various variants of mixup.</td>
<td>45</td>
</tr>
<tr>
<td>5.9</td>
<td>A comparison of the PolitBERT model trained with the three best-performing variants of mixup plus EDA.</td>
<td>46</td>
</tr>
<tr>
<td>5.10</td>
<td>A comparison of the PolitBERT model trained with various degrees of dropout probabilities.</td>
<td>46</td>
</tr>
<tr>
<td>5.11</td>
<td>A comparison between the domain-specifically pre-trained PolitBERT and the generally pre-trained BERT.</td>
<td>46</td>
</tr>
<tr>
<td>5.12</td>
<td>A comparison between different pre-trained language models trained on the multi-class classification task.</td>
<td>47</td>
</tr>
<tr>
<td>5.13</td>
<td>A comparison of the different ensembles of PolitBERT and the normal multi-class classifier.</td>
<td>47</td>
</tr>
<tr>
<td>A.1</td>
<td>The testset accuracies of the additional multi-class models split up by author.</td>
<td>58</td>
</tr>
<tr>
<td>A.2</td>
<td>The macro-averaged benchmarks of the additional models trained on the multi-class task and their amount of parameters.</td>
<td>58</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Plots of the ReLU and Sigmoid activation functions. ................................. 8
2.2 The original Transformer architecture where N is equal to 6. The output of the last (of the N) encoder is fed to all N decoders. Picture taken from [127]. ............................. 11

3.1 A screenshot from a video deepfake algorithm animating Sean Bean’s head to mimic my facial expression and movements using a pre-trained network provided on Google Colab[51]. 16
3.2 A visualization of the proposed network architecture of Yoon Kim. Picture taken from [75]. 19
3.3 A visualization of the BERT architecture for pre-training (left) and fine-tuning (right). Picture taken from [30]. .......................................................... 20

4.1 Caption for LOF ................................................................. 23
4.2 An example of the common format of a transcript and how it is processed by the webscraper. 25
4.3 The distribution of sentences per author in Set_7. ...................................... 27
4.4 The distribution of sentence lengths in Set_7. .......................................... 27

5.1 The confusion matrix of the distribution of misclassification per author. .......... 39
5.2 The amount of misclassified sentences grouped by the amount of wordpieces they are represented with and divided by the total amount of sentences with the same length. ... 40
5.3 The interpretation result of the sentence "And thank you, Michelle, for being here." - Originally from Donald Trump, classified as Barack Obama. .......................... 42
5.4 The interpretation result of the sentence "Do you believe that I would vote for Barack Obama?" - Originally from Donald Trump, classified as Bernie Sanders. ........... 42
5.5 The interpretation result of the sentence "As president, I will not allow us to fail in Afghanistan." - Originally from Hillary Clinton, classified as Barack Obama... ........ 43

A.1 The training loss of pre-training PolitBERT .............................................. 55
A.2 The evaluation loss of pre-training PolitBERT .......................................... 55
A.3 The learning rate schedule used to pre-train PolitBERT ............................ 55
Bibliography


BIBLIOGRAPHY


[100] Hua Qi, Qing Guo, Felix Juefei-Xu, Xiaofei Xie, Lei Ma, Wei Feng, Yang Liu, and Jianjun Zhao. DeepRhythm: Exposing deepfakes with attentional visual heartbeat rhythms. 2020.


BIBLIOGRAPHY


