Bachelor Thesis

Domain transfer across country, time and modality in multiclass-classification of political texts

Author

Nadja Sauter

Supervisor

Dr. Matthias Aßenmacher
Prof. Dr. Christian Heumann

Department of Statistics

Ludwig Maximilian University of Munich

Munich, 25th of October 2022

Abstract

Natural language processing (NLP) is a research field of Artificial Intelligence that explores how computers can achieve human-like language understanding. Due to the increased interaction between computers and human language (e.g. Google Search), this research area has become more and more important in the last twenty years. One common NLP task is text classification. In the scope of this thesis, four different supervised learning approaches are investigated to classify political texts into eight categories. The models TF-IDF + Logistic regression, Word2Vec + Long Short Term Memory Networks, Doc2Vec + Logistic regression and BERT are compared on two labeled data sets. Firstly, manifestos from political parties are used to train and evaluate the models (within-domain classification). Secondly, the modality of the data is changed from written text to verbal speeches of New Zealand politicians, which are then classified by the already fitted models into the eight categories (cross-domain classification). BERT achieves the best evaluation scores on both data sets and is used for further experiments across time and country. The results of the additional analysis show that BERT can be applied on future data with similar performance. Moreover, there seem to be differences between the manifestos of different countries of origin.

Contents

List of Figures

List of Tables

List of Abbreviations

NLP	Natural Language Processing
LR	logistic regression
NN	neural network
FNN	Feedforward Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory Networks
Bi-LSTM	Bidirectional Long Short Term Memory Network
TF-IDF	term frequency - inverse document frequency
CBOW	Continuous Bag-of-Words
PV-DM	Distributed Memory version of Paragraph Vector
PV-DBOW	Distributed Bag of Words version of Paragraph Vector
BERT	Bidirectional Encoder Representations from Transformers
MLM	masked language modelling
NSP	next sentence prediction
UK	United Kingdom
USA	United States
GPU	graphical processing units

1. Introduction

In most parliamentary democracies every time before an election a political party publishes their party manifesto. This program summarizes the policy priorities for the period following the election and works as a guide for voters (Suiter and Farrell, 2011). Additionally, the content of the manifestos forms the foundation for the government coalition process as it summarizes the political goals of each party. Analyzing these texts can be very interesting from a political science perspective. For instance, Janda et al. (1995) investigate the common assumption that political parties often try to change their images following a poor election result. Other researchers examine if parties learn from foreign successful parties (Böhmelt et al., 2016). Tavits and Letki (2009) and Tsebelis (1999) also investigate different interesting research questions based on political manifestos. Thereby, they use amongst others the database of the Manifesto Project[1], which received the American Political Science Association award (APSA) for the best data set in comparative politics. It covers programs of over 1000 parties from 1945 until today in over 50 countries on five continents (Lehmann, 2022). The database provides access to all these texts and even an additional content analysis. Coders from over 50 different countries split the documents in quasi-sentences and classified each in a coding scheme of 54 categories, which were further summarized in eight topics. The hand-annotation is very time intensive and requires training of the human coders to ensure reliability.

In the age of artificial intelligence, this thesis explores how Natural Language Processing (NLP) can be used to classify the quasi-sentences of the English-language manifestos of seven chosen countries into the eight topics of the Manifesto coding scheme. Therefore, different NLP methods, namely TF-IDF, Word2Vec, Doc2Vec and BERT, are used to process and subsequently classify the text data. In the following, the related work, data and underlying theory are introduced. Apart from elaborating the different NLP models in detail, classification and neural networks are explained as well, including the two applied models multinomial logistic regression and Long Short Term Memory

[1]https://manifesto-project.wzb.eu/information/documents/manifestoR

networks. After setting the theoretical foundation, the explained models are fitted on the Manifesto Project corpus. The predictive performance of each model is compared based on the classification of the manifestos and across domains on an external data set consisting of political speeches. Finally, the best model is used for further experiments across time and country.

2. Related work and data

This thesis comprises two main research goals. Firstly, four different methods are compared with regard to within-domain and cross-domain classification of political texts. A detailed explanation of the two kinds of classification follows in Section 3.1.1. A model suggested by Osnabrügge et al. (2021) is used as a starting point. Secondly, the best model is used for further experiments across time and country. For the analysis of the two research questions different data sets are used, which are explained in the following.

2.1. Original paper and data

This work is based on the paper "Cross-Domain Topic Classification for Political Texts" (Osnabrügge et al., 2021) which uses a supervised learning approach for the classification of political manifestos and speeches. The data and the code of their analysis can be found online on Code Ocean [1]. For their analysis, two labeled text data sets are used with the eight labels "freedom and democracy", "fabric of society", "economy", "political system", "welfare and quality of life", "social groups", "external relations" and "no topic". The first data set, the source corpus ($N_S = 115{,}410$), consists of English-language manifestos annotated by the Manifesto Project from 1984 until 2018 from the following seven countries: Australia, Canada, Ireland, New Zealand, South Africa, the United Kingdom (UK) and the United States (USA) (Krause et al., 2018). Each document is split into quasi-sentences and then categorized by a trained coder of the Manifesto Project. One quasi-sentence mostly equals one sentence. However, some long sentences contain several statements and for this reason are split into multiple quasi-sentences. The source data set is used for training and the within-classification. The second data set, the target corpus ($N_T = 4{,}165$), consists of English speeches delivered by members of the New Zealand Parliament from 1987 to 2002. Osnabrügge et al. (2021) extracted the speeches from the official record of the New Zealand Parliament, the Hansard, and hand annotated 4,165. This data is used for the cross-domain classification.

[1] https://codeocean.com/capsule/0078777/tree/v1

In the paper, a classifier is trained on the Manifesto Project corpus to learn the eight topics of the coding scheme. Additionally, the authors create an own coding scheme with 44 topics. For text processing, they use the *scikit-learn* implementation of the TF-IDF vectorizer (Pedregosa et al., 2011) and then train a multinomial logistic regression classifier for the 8 and 44 topics respectively. After the hyperparameter tuning of the logistic regression with Grid Search, they use this classifier on the the held-out set of the manifestos and additionally apply it on the speeches. The classifier achieves an accuracy of 0.641 on the test set of the source corpus and an accuracy of 0.507 on the speeches for the eight topics, showing that cross-domain classification is a reasonable approach. The recall per category varies per topic. For instance, the category "no topic" is mostly misclassified, whereas "welfare and quality of life" has within and across domains the highest recall. To illustrate the usefulness of their method, they apply it to two empirical use cases to investigate how electoral reforms and the gender of parliamentarians affect the topics of the speeches.

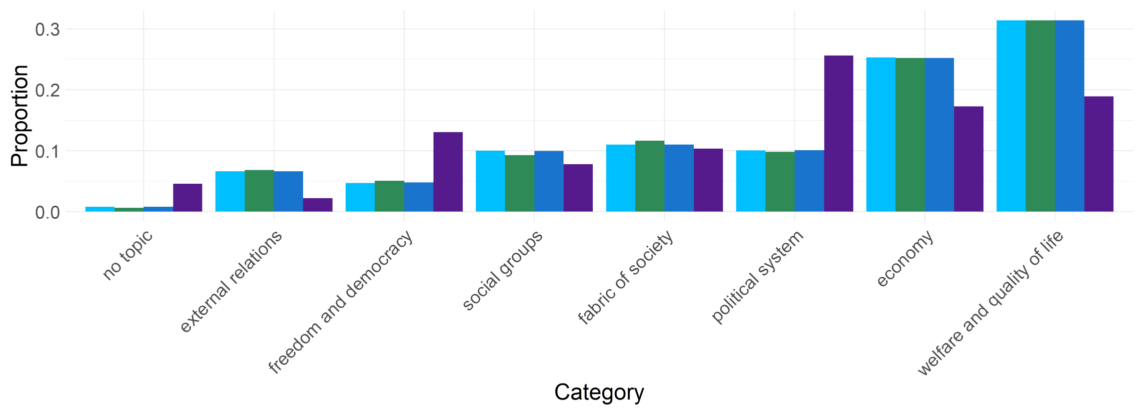
The first research goal of this thesis, which compares the different modelling approaches, uses the two data sets provided by the introduced paper. The TF-IDF classifier of the paper is reproduced with minor changes and then compared to the other models (see Subsection 4.2). The goal is to investigate if another method can classify the text better than the approach of Osnabrügge et al. (2021).

2.2. Data extraction from Manifesto Project

For further experiments across time and country, the manifestos are extracted independently via the R-package *manifestoR* from the Manifesto Project, including additional information about the publishing year and country of origin. This information is not given in the source data set of the paper. The same corpus version (2018-2) of the paper, including data until 2018, and the updated version (2021-1) until the year 2021 are extracted, resulting in $N_{2018} = 114{,}523$ and $N_{2021} = 151{,}807$ observations respectively. Thereby, a difference of 887 text examples between the 2018 version of the paper ($N_S = 115{,}410$) and the manually generated data set ($N_{2018} = 114{,}523$) can be observed, which is probably due to changes in the database of the Manifesto Project.

In total, four different data sets are used for the different research questions. Figure

2.1 shows the distribution of the eight categories for each data set. All versions of the source corpus follow the same distribution with the categories "welfare and quality of life" and "economy" representing together about 57% of the data. On the other hand, the most common class of the target corpus is "political system" with a portion of 26% followed by "welfare and quality of life" with 19%. Thus, the frequency per category between the target and source corpus is different. Overall, one can see that the classes are imbalanced in all data sets.



	Data	Number of observations	Description
🟦	Source data 2018	N_{2018} = 114, 523	Extracted manifestos from Manifesto Project (version 2018)
🟩	Source data 2021	N_{2021} = 151,807	Extracted manifestos from Manifesto Project (version 2021)
🟦	Source data paper	N_S = 115,410	Data from paper comprinsing manifestos (version 2018)
🟪	Target data paper	N_T = 4,165	Data from paper comprising the speeches from New Zealand

Figure 2.1.: Distribution of the eight categories for the different data sets. The legend includes detailed descriptions of the four data sets. It can be distinguished between the two extracted data sets and the two data sets from the paper (Osnabrügge et al., 2021).

3. Methodology

This chapter introduces the methodological background for the following application of political text classification. Firstly, some fundamental knowledge about classification and neural networks is explained. Afterwards the methods TF-IDF, Word2Vec and Doc2Vec are presented for feature engineering in NLP. These methods are applied later to create numeric representations of the text as input for the classifier. Lastly, the theory behind the state-of-the-art NLP model BERT is elaborated.

3.1. Classification

In classification tasks, data is categorized into different groups by supervised or unsupervised learning (Solomon and Breckon, 2011). The latter approach is used for unlabeled data, whereas supervised learning is applied in case of labeled data. Since every text example is assigned to exactly one category in the data of the implementation (see Section 2), supervised learning is utilized for the multi-class text classification task. It can be further distinguished between within-domain and cross-domain classification, which is explained in the following. Moreover, the theory about the common classifier logistic regression is elaborated and later applied in the experiments. In order to evaluate the results of a classifier, different evaluation metrics are introduced.

3.1.1. Within-domain and cross-domain classification

In within-domain classification, the model is tested on data from the same origin as the training data. On the other hand, in cross-domain classification, data is classified which is similar to the training data, but from a different source (Burscher et al., 2015). In the following experiments, the different models are trained on the source corpus, which consists of the political party manifestos. Using the trained model on a held-out-set of the source corpus is within-domain classification, whereas classifying the speeches of the target corpus is cross-domain classification. While the speeches are also of political context, the modality differs between verbal and written. Thus, the model is used across domains, which can also be seen as a form of transfer learning. Instead of retraining a new model from scratch, it is possible to reuse an already existing one, which saves time and effort. Frequently, the labels of the target corpus are not given in the data set. As obtaining this additional information can be time and resource intensive or even impossible, it is often suitable to apply cross-domain classification. To achieve good performance across domains, the data sets need to be similar enough, so that the classifier can identify patterns in the target data which it learned on the source corpus. In order to measure the success of the transfer across domains, the target corpus was labeled by Osnabrügge et al. (2021) for the following experiments. This is mostly not given in practice and rather for research purposes here.

3.1.2. Logistic regression

One of the most popular classifiers is logistic regression (LR), which predicts the probabilities of a binary, dependent variable y, given a set of independent variables $x \in R^n$ (Fahrmeir et al., 2021). Following the notation of the *scikit-learn* documentation of the binary LR with $y \in \{0, 1\}$, $P(y_i = 1 \mid X_i)$ is predicted as

$$\hat{p}(X_i) = \operatorname{expit}(X_i w + w_0) = \frac{1}{1 + \exp(-X_i w - w_0)} \tag{3.1}$$

with $w \in R^n$ as weight vector (Pedregosa et al., 2011). LR minimizes the following cost function

$$\min_{w} C \sum_{i=1}^{n} \left(-y_i \log(\hat{p}(X_i)) - (1 - y_i) \log(1 - \hat{p}(X_i))\right) + r(w) \tag{3.2}$$

where $r(w)$ represents the regularization term and $C \in \mathbb{R}^+$ is the inverse of the regularization strength. It is common to use regularization to prevent overfitting and to handle high dimensional data better (Wang et al., 2008). A widely used approach is L2-regularization, which is also referred to as Ridge regression with $r(w) = \frac{1}{2}\|w\|_2^2 = \frac{1}{2}w^T w$ (Fahrmeir et al., 2021).

Extending the binary case to a K-class classification problem with $y_i \in 1, \ldots, K$, the estimated class probabilities $P(y_i = k \mid X_i)$ can be formulized as

$$\hat{p}_k(X_i) = \frac{\exp(X_i W_k + W_{0,k})}{\sum_{l=0}^{K-1} \exp(X_i W_l + W_{0,l})} \tag{3.3}$$

with W as a matrix of coefficients (Pedregosa et al., 2011). The regularized objective function for the optimization becomes

$$\min_W -C \sum_{i=1}^{n} \sum_{k=0}^{K-1} [y_i = k] \log \left(\hat{p}_k \left(X_i\right)\right) + r(W) \tag{3.4}$$

where $[y_i = k]$ represents the Iverson bracket which evaluates to 0 if the statement is false and to 1 otherwise. The minimization of the LR objective can be solved with several optimization algorithms such as SAGA (Defazio et al., 2014) or Newton-CG algorithm (Royer et al., 2020). Moreover, LR is part of the generalized linear models and also called the Maximum Entropy model (ME) in the Natural Language Processing field (Solomon and Breckon, 2011; Dobson and Barnett, 2018).

3.1.3. Evaluation metrics for classification

In order to assess the performance of a classifier, some measurement needs to be defined, which is also known as evaluation metric. The most intuitive one is accuracy which determines how often the model predicts the correct label.

$$\text{Accuracy } = \frac{\text{Number of correct predictions}}{\text{Number of observations}} \tag{3.5}$$

However, this form of measurement does not consider if a lot of observations are misclassified in a special way, so that there are a many false negatives or false positives. Therefore, other measurements such as recall and precision should be considered. Recall describes the proportion of correctly identified positives, whereas precision quantifies what proportion of positive identifications is actually correct (Xiao and Sun, 2021):

$$\text{Recall } = \frac{TP}{TP + FN} \quad \text{Precision } = \frac{TP}{TP + FP} \tag{3.6}$$

with TP = True Positives, FP = False Positives and FN = False Negatives. The F1 score takes both precision and recall into account (Xiao and Sun, 2021).

$$F_1 = \frac{2 \cdot \text{ Precision } \cdot \text{ Recall}}{\text{Precision } + \text{ Recall}} \tag{3.7}$$

In order to calculate the F1 score in multi-class classification, the way of aggregating the F1 scores per class needs to be defined. The micro F1 takes the global number of TP, FN and FP into account and is equal to accuracy in the multi-class case. The weighted

F1 calculates the individual F1 scores per class and takes the weighted sum, where the weight w depends on the number of observations per class (Pedregosa et al., 2011).

$$w_1 F1_{\text{class 1}} + w_2 F1_{\text{class 2}} + \cdots + w_K F1_{\text{class } K} \text{ with } K = \text{number of classes} \tag{3.8}$$

In this way, the weighted F1 favours the majority class. In comparison, the macro F1 does not use any weights, resulting in a bigger penalisation for the misclassification of minority classes (Xiao and Sun, 2021):

$$\frac{1}{K}(F1_{\text{class 1}} + F1_{\text{class 2}} + \cdots + F1_{\text{class } K}) \tag{3.9}$$

Overall, evaluation scores summarize the performance of a model into one measure, focusing on different kinds of misclassifications. In order to get more details about the wrong predictions per class, a confusion matrix can be very helpful. This is a $K \times K$ matrix comparing the true labels of each observation with the predictions of the model.

3.2. Neural network

A neural network (NN) is a computing system inspired by the functioning of the human brain. The working units of the brain are neurons, which are connected to each other and transmit information over electrical impulses when some threshold is exceeded. The first artificial, single neuron model, also called perceptron, was developed by Frank Rosenblatt (1958). The idea was further developed to multi-neuron networks, but research in this area stagnated in the 1960s due to the lack of computational power to train these complex models. This changed in recent decades with the advent of cloud computing and graphical processing units (GPU), which made it possible to train neural networks efficiently and fast. Another beneficial development is that larger and larger data sets are available nowadays which is important for the training procedure as well. Today, neural networks are a state-of-the-art supervised learning method for many tasks. The fundamental architecture is the so-called Feedforward Neural Network (FNN) (see Section 3.2.1). Another common form is the Recurrent Neural Network (RNN) for sequential data (see Section 3.2.2) or Convolutional Neural Networks which are especially used in Computer Vision.

3.2.1. Feedforward neural network

A FNN consists of three parts: The input layer takes the data and passes it through an arbitrary number of so-called hidden layers to the final output in the last layer. The information only flows in one direction from the input to the output layer, explaining the term feedforward. In Figure 3.1 the circles represent the neurons which are connected to each other, controlling the influence of the previous layer by weighting their input and transforming it with a non-linear function. The illustration shows a fully connected FNN as all neurons are connected to every neuron in the previous layer. Here, the input is a vector $x = (x_1, x_2, x_3, x_4, x_5, x_6)$ with 6 dimensions which is passed through two hidden layers of width 4 and 3, describing the number of units per layer. As the depicted architecture has more than one hidden layer, this is also called a deep learning neural network of depth = 2 (Goodfellow et al., 2016). Thus, the depth of a FNN equals the number of hidden layers.

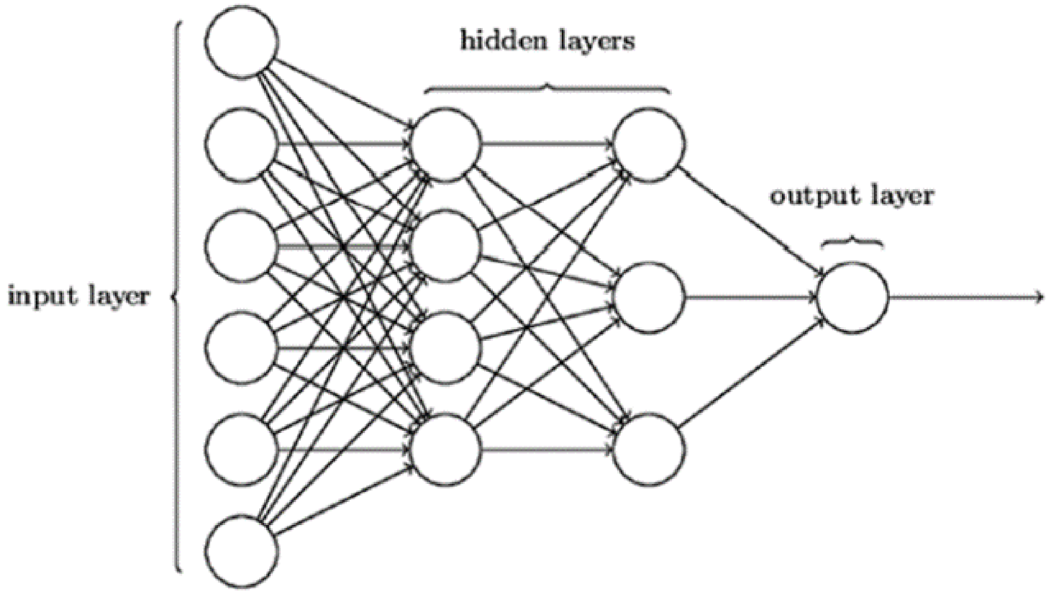


Figure 3.1.: Simple Feedforward Neural Network with two hidden layers (Goyal et al., 2018)

FNNs can be described mathematically as a parameterized function, mapping the input x to the output $\hat{y} = f(x; \theta)$ (Goodfellow et al., 2016). The goal is to approximate the function f by learning the most suitable parameters θ, following the idea of the maximum likelihood estimation. It can be shown that a sufficiently complex neural network is able to learn almost any function in any dimension as universal approximator (Hornik et al., 1989). This can be achieved by nesting functions

$$f(x;\theta) = h^{(2)}\left(h^{(1)}(x;\theta)\right) \tag{3.10}$$

where $h^{(1)}$ represents the first layer and $h^{(2)}$ the second one. More precisely, each h can be defined as $h = g\left(W^{\top}x + b\right)$ with W as a weight matrix, x as input and b as intercept (Goodfellow et al., 2016). For a FNN of depth 2, as depicted in Figure 3.1, the following chain structure can be formulized, where the first layer is given by

$$h^{(1)} = g^{(1)}\left(W^{(1)\top}x + b^{(1)}\right) \tag{3.11}$$

and the second layer defined as

$$h^{(2)} = g^{(2)}\left(W^{(2)\top}h^{(1)} + b^{(2)}\right). \tag{3.12}$$

The function g is called activation function and is a non-linear mapping in order to represent non-linearities in the data. Typical activation functions are sigmoid, tanh or the Rectified Linear Unit (ReLU). Moreover, the activation function has to be differentiable, so that gradient-based methods can be used for training. A loss function is defined as $\mathcal{L}\left(\hat{y}, y\right) = \mathcal{L}\left(f\left(x;\theta\right), y\right)$ (Goldberg, 2017) to measure how well the current parametrization approximates the mapping f by comparing the actual values y with the predictions $\hat{y}$ of the model. In order to minimize the loss via gradient descent and backpropagation, the partial derivatives are calculated iteratively and used to adjust the weights.

In order to avoid overfitting, different regularization methods are usually used. One common approach is to add a regularization constraint on the parameter values in the loss (e.g. L2-regularization, see Section 3.1.2). Another option is to implement drop out or early stopping (Goldberg, 2017). The first method skips connections randomly during training, so that the FNN is not fully connected. The latter stops training when no further improvement can be seen on the validation set measured by the loss or some evaluation metric. This can reduce the generalization error, which describes the performance of the model when introduced to unseen data (Larsen and Hansen, 1994).

Another important part of fitting a FNN is hyperparameter tuning. The goal is to maximize the performance of a model by adjusting the parameters of the model such as the number of hidden layers or the proportion of dropout (Andonie, 2019). Grid search is one applicable algorithm to perform hyperparameter optimization in an exhaustive manner over all combinations of the specified hypererparmeter space (Liashchynskyi and Liashchynskyi, 2019).

3.2.2. Recurrent neural network

The idea of RNNs was first investigated by Rumelhart et al. (1985) as a special form of neural networks used for sequential or temporal data of tasks such as speech recognition, language translation or time series. In contrast to FNNs, this architecture takes information from prior elements into account that can influence the current output, acting as a memory of the RNN. As depicted in illustration 3.2, RNNs can be visualized as a chain of multiple copies of the same network with one single hidden layer. The method keeps the order of the input x_t at time step t. The hidden states h can be mathematically formulated as the recurrence relation (Goyal et al., 2018)

$$h_t = f_1 \left(W_{hh} h_{t-1} + U_{xh} x_t \right) \tag{3.13}$$

where W_{hh} is a mapping across hidden states, U_{xh} is a mapping from input x to the hidden layer h and f_1 is applied element-wise and is usually a tanh or sigmoid function. The first term is based on the previous hidden state, whereas the second one takes the current input into account. For example, this structure allows RNNs to capture the sequential relationships between words of a sentence. Nevertheless, they struggle with long-term dependencies, mathematically resulting in vanishing or exploding gradients because of the recursive multiplication of the weights. This problem is tackled by LSTM, which are a special type of RNNs.

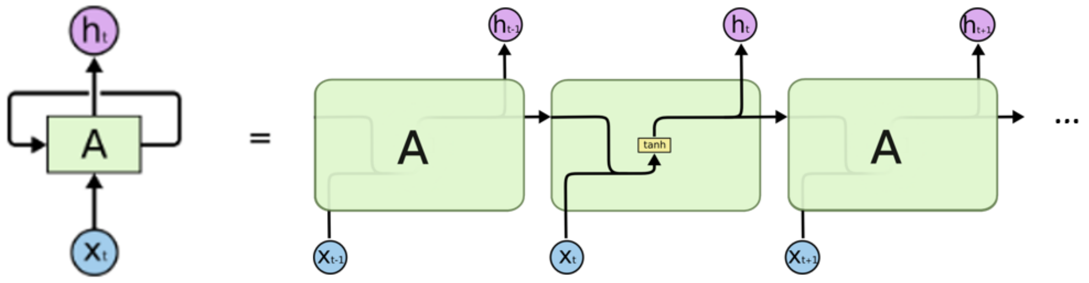


Figure 3.2.: RNN in rolled and unrolled illustration (Olah, 2015)

LSTMs were first introduced by Hochreiter and Schmidhuber (1997) and solve retaining information over longer time periods. This is achieved by implementing additional gates, which control the influence of the last cell state and the new input on the new cell state. Like RNNs, LSTMs also have the chain-like structure, but each unit has a more complex composition (see Figure 3.3). The repeating module consists of four network layers, interacting in a special way through three gates. Taking the same notation and illustration as Goyal et al. (2018), x_t represents the input in the following, C_t denotes the cell state, h_t represents the hidden state, b_t denotes the bias and W_t denotes the weights. This is always relative to the corresponding layer and time step t.

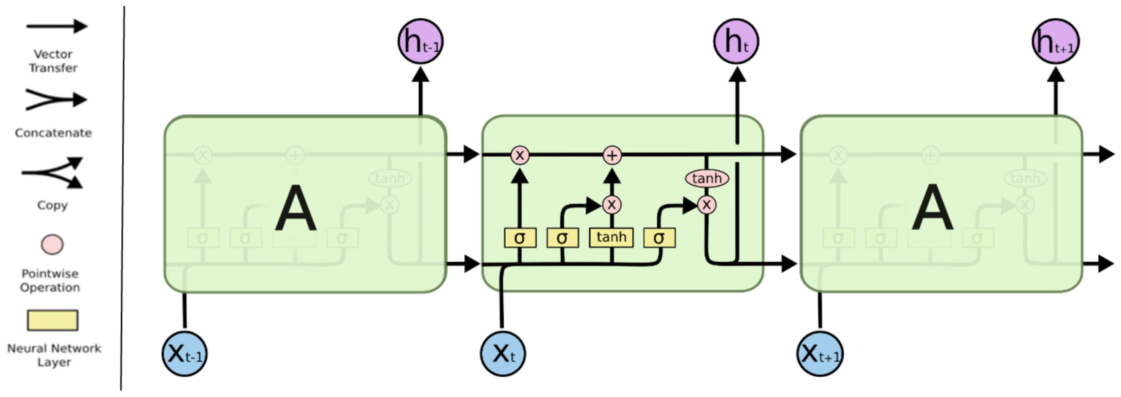
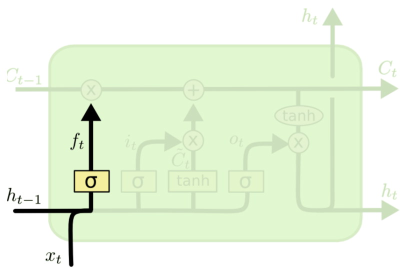


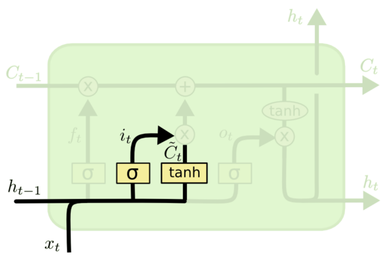
Figure 3.3.: Architecture of a LSTM (Goyal et al., 2018)

Firstly, the **forget gate** controls to which extent the previous hidden state h_{t-1} and the input x_t influences the current cell state.



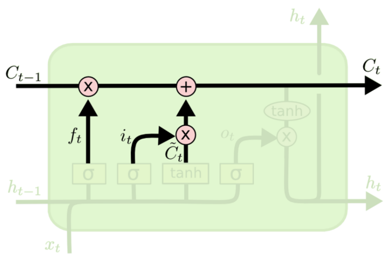
$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right) \qquad (3.14)$$

The **input gate** influences the contribution of the new input i_t to the memory by deciding which values will be updated via a sigmoid function, whereas the tanh layer creates a vector of new candidate values $\dot{C}_t$.



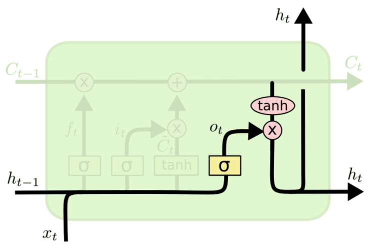
$$i_t = \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right)$$
$$\dot{C}_t = \tanh\left(W_C\left[h_{t-1}, x_t\right] + b_c\right) \tag{3.15}$$

The horizontal line in the top of the figure illustrates the cell state C_t, which will be updated pointwise in the next step based on the forget gate f_t and input gate i_t as well as the previous cell state C_{t-1} and the candidate values $\dot{C}_t$.



$$C_t = f_t \cdot C_{t-1} + i_t \cdot \dot{C}_t \tag{3.16}$$

Finally, through a sigmoid layer the output o_t of the **output gate** is calculated based on the hidden state h_{t-1} and the input x_t. Then the cell state is put through a tanh activation function and multiplied pointwise by o_t for the final output h_t.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t \cdot \tanh\left(C_t\right) \tag{3.17}$$

This is the general functioning of a LSTM unit. Different modifications exist such as the Bidirectional Long Short Term Memory Network (Bi-LSTM). Here, two individual hidden layers are trained and coupled to the same output layer (Schuster and Paliwal, 1997; Zhu et al., 2015). The first layer processes the sequence forward, whereas the second layer processes the information backwards.

3.2.3. Transformer

The transformer is a deep learning model, consisting of an encoder-decoder architecture (see Figure 3.4). The encoder learns a latent representation of the input which is then used by the decoder to generate the desired output (Vaswani et al., 2017). It is possible to stack several encoders and decoders on top of each other.

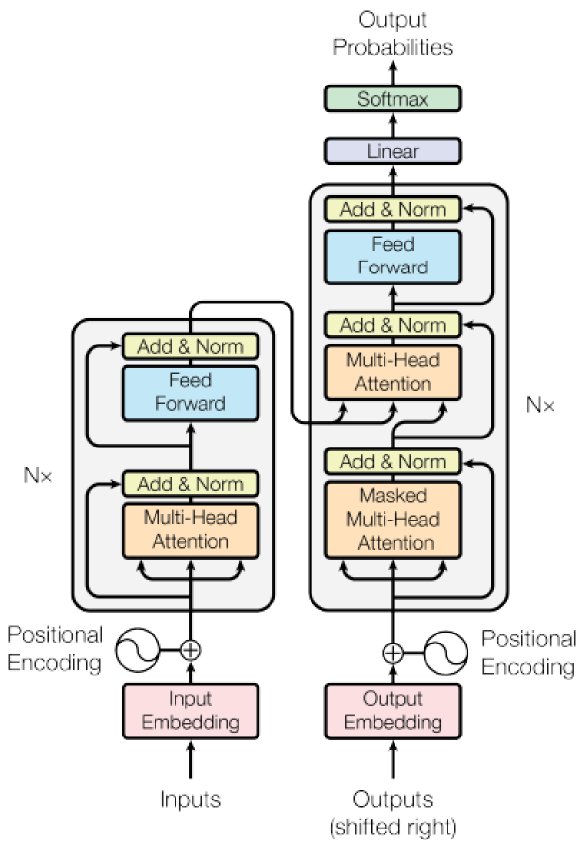


Figure 3.4.: Architecture of the transformer (Vaswani et al., 2017)

The special mechanism used by transformers is called attention, which was first introduced by Bahdanau et al. (2014). This method enables the model to capture long-term dependencies without any recurrence and sequential computation as in RNNs (see Subsection 3.2.2). The input can be processed in parallel, reducing computing time tremendously. This advantage makes the transformer an important architecture for several tasks in NLP as well as Computer Vision. The implemented scaled dot-product attention (Vaswani et al., 2017, see Figure 3.5 on the left side) is defined as the softmax of the dot product of a query vector Q and a key vector K scaled by their common dimension d_k and then multiplied by the value vector V of dimension d_v as formulated in the following in matrix notation.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.18}$$

This form of attention captures associations between words and can handle long term dependencies. In this way, shortcut connections between the different inputs that relate to each other are created. Instead of calculating only one attention matrix, the transformer concatenates h attention matrices, also called multi-head attention (Vaswani et al., 2017, see Figure 3.5 on the right side).

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}\left(\text{head}_1, \ldots, \ \text{head}_h\right) W^O \\ \text{where head}_i &= \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{3.19}$$

with the parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ with d_{model} set to the output dimension of the embedding layers. In this way the transformer is able to consider multiple relationships between words.

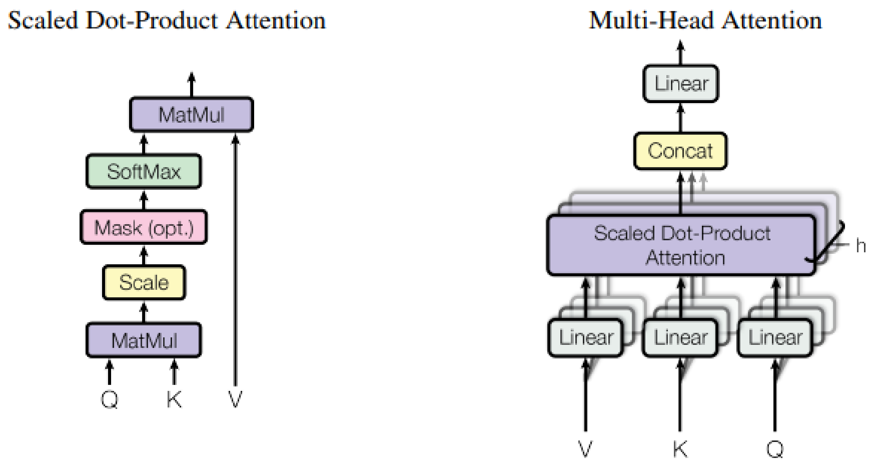


Figure 3.5.: Illustration of scaled dot-product attention (left) and multi-head attention (right) (Vaswani et al., 2017).

3.3. Feature engineering in NLP

NLP is a research field of Artificial Intelligence that explores different methods about how computers can achieve human-like language understanding (Liddy, 2001). To achieve this goal, the computer needs to learn the grammatical rules and the semantic meaning of words. Before working with the text data, it first needs to be processed into some numerical representation. In order to achieve this form of feature engineering, different NLP methods can be applied.

3.3.1. TF-IDF

The term frequency - inverse document frequency (TF-IDF) is a simple count-based approach, which encodes the vocabulary in a matrix according to the word frequencies in the corpus. It consists of two parts. Using the notation of Karabiber et al. (2022), the term frequency (TF) measures how often a word occurs in one document weighted by the length of the document.

$$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}} \tag{3.20}$$

The second part, the inverse document frequency (IDF), calculates how important a word is by taking the logarithmized ratio of the total number of documents and the number of documents including the word.

$$IDF = \log\left(\frac{\text{number of documents in the corpus}}{\text{number of documents in the corpus that contain the term}}\right) \tag{3.21}$$

The TF-IDF score then results from the multiplication of both values.

$$\text{TF-IDF} = TF \cdot IDF \tag{3.22}$$

The experiments in this thesis are conducted with the TF-IDF vectorizer of *scikit-learn* (Pedregosa et al., 2011). Here, the TF is not normalized and the IDF is slightly modified:

$$\begin{gathered} TF = \text{ number of times the term appears in the document} \\ \text{IDF}(t) = \log\frac{1+n}{1+\text{df}(t)} + 1 \end{gathered} \tag{3.23}$$

where n is the total number of documents in the corpus, and $\text{df}(t)$ denotes the number of documents that contain term t. Adding one to the numerator and denominator avoids zero division as if an extra document was seen, containing every word in the collection exactly once. The resulting TF-IDF vector per document $v = TF \cdot \mathrm{IDF}(t)$ is normalized by the Euclidean norm

$$v_{\text{norm}} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_n}^2}}. \tag{3.24}$$

The TF-IDF can be interpreted as a measurement of word importance. On the one hand, TF measures commonality within a document, whereas IDF incorporates rare words between documents (Karabiber et al., 2022). This provides a score of how important a word is rated within a document relative to a corpus. Thus, the importance is high when a word appears often in a given document and rarely in others.

As a result of applying TF-IDF over the whole vocabulary space on every document, a document-term matrix is constructed of shape (number of documents $\times$ vocabulary size). Every row corresponds to one document, each column to one word and in this way each cell denotes the TF-IDF value. Salton et al. (1975) first utilized this method to measure the semantic similarity of document pairs, as similar documents have similar TF-IDF values of corresponding words. Consequently, similar rows of the document-term matrix can be expected for similar documents. This approach is document centric, meaning that one representation is created for every document, which can be used as input for a classifier (Pilehvar and Camacho-Collados, 2020). For instance, in the following classification of political texts, each row of the document-term matrix represents one text example, which is taken as input for the logistic regression classifier (see Section 4.2). The method results in a sparse representation of the documents, meaning that the document-term-matrix mostly consists of zero entries as the number of columns of the matrix are given by the vocabulary size of the whole corpus. Moreover, this method is only based on word frequencies, so that the context as well as the syntactic and semantic relationships of words are not considered. These characteristics of language understanding are taken into account in the following approaches.

3.3.2. Word2Vec

A more advanced method is Word2Vec (Mikolov et al., 2013b) which is based on the distributional hypothesis that “a word is characterized by the company it keeps” (Firth, 1957), implying that words that occur in the same context are likely to have a similar meaning as well (Harris et al., 1954). For instance, the words "congress" and "government" have likewise semantics, so that they appear with similar surrounding words (e.g. congress/ government passed a law). By evaluating the context, words that occur with similar words will be given a similar representation. This can be achieved by creating so-called word embeddings, which are representations of words as continuous vectors in a multidimensional vector space, that mathematically describes the vocabulary space. Natural language modelling (Bengio et al., 2000; Collobert et al., 2011) uses neural networks to learn these dense word representations by a self-supervised next word prediction task, so that no labeled data is needed for training. Word2Vec (Mikolov et al., 2013b) is a predictive model based on a simple feedforward neural network architecture (see Subsection 3.2.1). There are two different procedures to create the embeddings, namely Continuous Bag-of-Words (CBOW) and Skip-gram. The first method infers the current word based on the surrounding words of a defined window, whereas the latter model works the other way round and predicts the context words given the target word. The different architectures are illustrated in Figure 3.6. Both models consist of input, projection and output layer and are trained via stochastic gradient descent and back-propagation (Rumelhart et al., 1986). However, according to the different training task, the objective functions are different.

Given the target word w_t and its surrounding words $w_{t-k}, \ldots, w_{t+k}$, the objective of the CBOW is defined as (Le and Mikolov, 2014)

$$\frac{1}{T} \sum_{t=k}^{T-k} \log p\left(w_t \mid w_{t-k}, \ldots, w_{t+k}\right) \tag{3.25}$$

based on the softmax

$$p\left(w_t \mid w_{t-k}, \ldots, w_{t+k}\right) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{3.26}$$

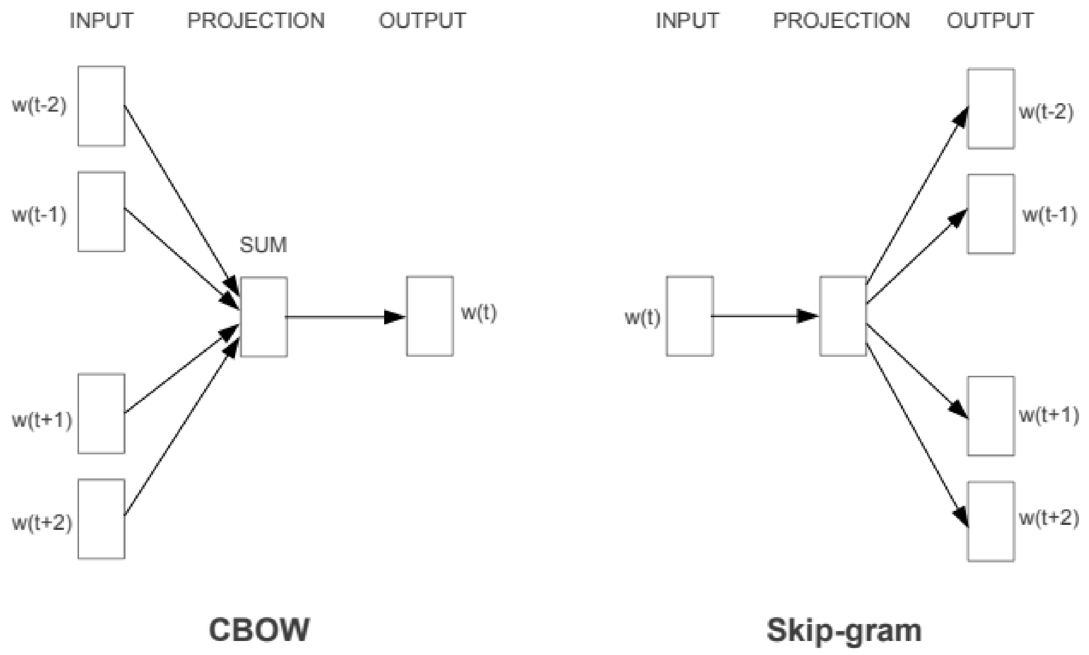


Figure 3.6.: Architecture of CBOW and Skip-gram (Mikolov et al., 2013a).

On the other hand, given a sequence of training words $w_1, w_2, w_3, \ldots, w_T$ Skip-gram maximizes the average log probability of the context words w_{t+j} given the target word w_t and the window size c (Mikolov et al., 2013b)

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p\left(w_{t+j} \mid w_t\right). \tag{3.27}$$

Thereby, $p\left(w_{t+j} \mid w_t\right)$ is originally defined using the softmax function (Mikolov et al., 2013b)

$$p\left(w_O \mid w_I\right) = \frac{\exp\left({v_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top} v_{w_I}\right)} \tag{3.28}$$

where W denotes the number of words in the vocabulary and v_w and v'_w are the input and output vector representations of w.

In practice, a hierarchical softmax (Morin and Bengio, 2005) with the structure of a binary Huffman tree is preferred to the standard softmax function for faster training in both cases. Another common option is negative sampling (Mikolov et al., 2013b) based on the idea of Noise Contrastive Estimation (NCE) (Gutmann and Hyvärinen, 2012). In this case, the Skip-Gram model learns to differentiate the true context words from randomly generated noisy words by means of logistic regression. After training, the word embeddings can be extracted as a weight matrix of the projection layer and used for further analysis like text classification. The vector size of the word representation equals the number of hidden units of the projection layer.

The resulting embeddings are static (Jurafsky and Martin, 2022) which means that one fixed vector is learned for each word. Furthermore, the embeddings are able to capture semantic relationships between words, meaning that a word embedding is close to semantically similar words in space, whereas dissimilar words are far from each other. Thereby, the similarity between embeddings can be mathematically described by vector operations such as the euclidean or cosine distance. It is even possible to meaningfully apply vector addition and subtraction. For instance, the vector("Queen") can be approximated by vector("King") - vector("Man") + vector("Woman") (Mikolov et al., 2013a).

3.3.3. Doc2Vec

Instead of learning word embeddings, it is also possible to create a representation of a whole document as it is done by Doc2Vec. This method is a self-supervised machine learning algorithm used to convert a document into a low-dimensional vector representation, which was originally introduced as paragraph vector by Le and Mikolov (2014). The method follows a similar approach as Word2Vec (see Section 3.3.2), so that the neural network architecture is identical apart from an added paragraph matrix and paragraph ID which identifies each document (see Figure 3.7).

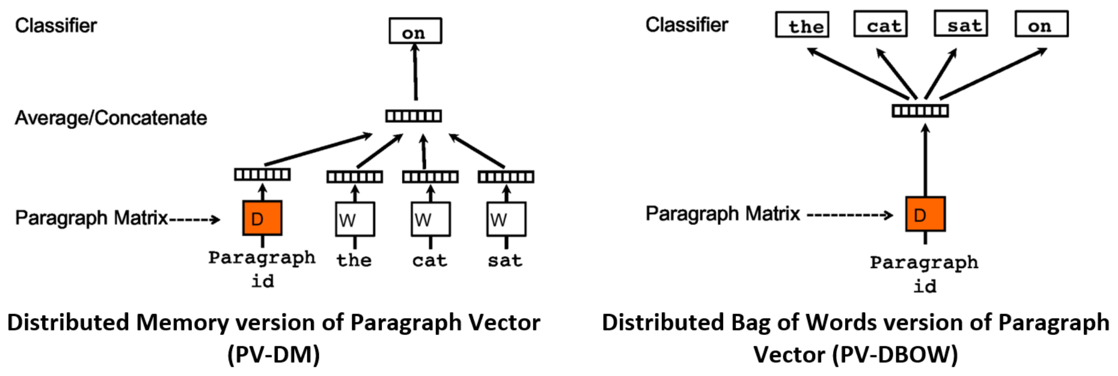


Figure 3.7.: Architectures of Doc2Vec (Le and Mikolov, 2014).

After training, the paragraph matrix can be extracted as a dense document representation. Similar to the CBOW method of Word2Vec, the Distributed Memory version of Paragraph Vector (PV-DM) predicts a target word based on the surrounding words in a specified text window and the additional paragraph ID, which are either averaged or concatenated by a parse tree of a sentence (Socher et al., 2011), to predict the next word. On the other hand, the Skip-gram analogy, called Distributed Bag of Words version of Paragraph Vector (PV-DBOW), uses the paragraph ID as input for predicting words of a randomly sampled text window of the corresponding document. PV-DBOW (extension of Skip-Gram) takes less computational time, whereas in the Word2Vec approach the CBOW method is faster. Following the same approach as Word2Vec, the paragraph vectors of Doc2Vec have the same characteristics on a document level (e.g. calculating distances as similarity measurement, see the last paragraph of Section 3.3.2). Thus, the resulting row vectors of the paragraph matrix can be used to calculate similarities between documents by taking the cosine distance or as input of a classifier.

3.4. BERT

The Bidirectional Encoder Representations from Transformers (BERT) is a general-purpose language model (Devlin et al., 2018), which was a major breakthrough in the field of NLP with great results in different language tasks such as text generation, question answering and sentence classification. BERT's model architecture is a deep bidirectional transformer encoder (see Section 3.2.3), which means that it learns deep representations of words by considering both, left and right context (Esposito et al., 2022). Unlike Word2Vec, BERT learns dynamic contextual embeddings, so that representations for the same word differ based on the context. This makes it possible to tackle ambiguity between words and handle polysemes, which are words with different meanings such as "pupil" (e.g. pupil as part of the eye and pupil as a synonym for student).

There are two steps involved in using BERT: pre-training and fine-tuning. The first part incorporates two self-supervised learning tasks, namely masked language modelling (MLM) and next sentence prediction (NSP). These are applied on a huge unlabeled corpus, consisting of the BooksCorpus with 800M words (Zhu et al., 2015) and English Wikipedia with 2,500M words (Devlin et al., 2018). The first task masks randomly 15% of the words by replacing them with a [MASK] token. Then the network is trained to predict these hidden words of the sentence. The second task, NSP, comprises binary classification, which predicts whether two sentences are following each other in the corpus or not. After pre-training, the model is fine-tuned on a specific data set and task via supervised learning. This approach is also known as a downstream task and is part of transfer learning. The big advantage of this procedure is that time and resources can be saved by just reusing an already existing model instead of fitting a new one from scratch. Especially as language models are very big, applying a pre-trained model is a huge benefit. There are different ways of fine-tuning the pre-trained model. Mostly, all weights are adjusted during fine-tuning (e.g. default of Hugging Face implementation). However, it is also possible to freeze some layers and fine-tune only the selected layers, which are usually the task related stacked head layers.

Before training, the input data needs to be tokenized into three embeddings as can be seen in Figure 3.8. The token embeddings are created by adding the [CLS] token at the beginning of the first sentences and the [SEP] token as separator of each sentence. The values of the token embeddings are learned during training. The [CLS] token holds the aggregated representation of the whole sentence, so that solely this special token is passed to the classifier for sequence classification. In the original paper, Devlin et al. (2018) add a single-layer network and a softmax as classifier. Furthermore, the segment embedding is used to distinguish between sentences. As the transformer is trained in parallel and not sequentially, the third part, the position embedding, enumerates the tokens to keep track of the word order. In this process, the WordPiece tokenizer (Wu et al., 2016) is used to split each sentence into tokens. If a word is not present in the vocabulary, the word is split until it is part of the vocabulary or separated in individual characters. This is effective in handling out-of-vocabulary words. For instance, in Figure 3.8 the word "playing" is separated into "play" and "##ing". Overall, BERT comprises a vocabulary of 30K tokens.

Depending on the configuration of BERT, the number of self-attention heads A (see Section 3.2.3 multi-head attention), the number of encoder layers L and the number of hidden units H differ. The two major architectures are $BERT_{BASE}$ with $L = 12, A = 12, H = 768$ and $BERT_{LARGE}$ with $L = 24, A = 16, H = 1024$ (Devlin et al., 2018). $BERT_{LARGE}$ represents a more complex architecture with 340 million parameters compared to $BERT_{BASE}$ with 110 million parameters. There are further versions of BERT such as $M - BERT$ for different languages apart from English or $DistilBERT$. The latter is a slimmed-down version of $BERT_{BASE}$ published by Hugging Face (Sanh et al., 2019). They reduced the size to 40% of the original model, while retaining 97% of its language understanding capabilities and being 60% faster in their experiments. The compression technique of knowledge distillation (Buciluă et al., 2006; Hinton et al., 2015) is used for pre-training. Therefore, the student model $DistilBERT$ is trained on the target probabilities of the original teacher model $BERT_{BASE}$. Thus, the student learns to make the same predictions as the teacher by matching the output distribution. Adding a cosine embedding loss to the objective function is suggested to better align the hidden state vectors of the two models (Sanh et al., 2019).

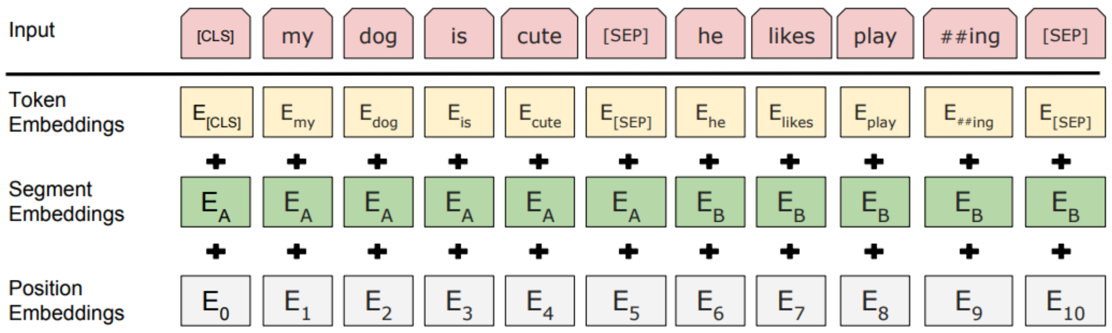
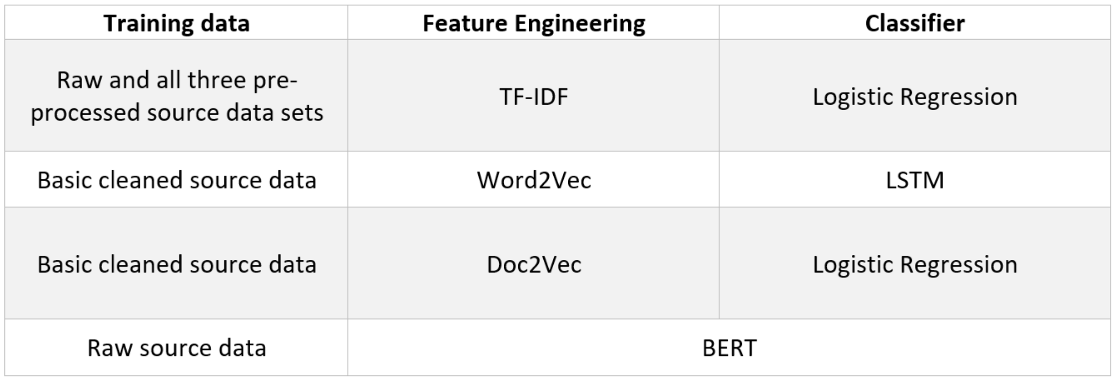


Figure 3.8.: Input representation of BERT (Devlin et al., 2018).

4. Implementation of NLP models

In this chapter, the introduced methods are used to train different classifiers on the manifestos and apply them across domains on the speeches (see overview in Table 4.1). After some pre-processing, the NLP methods TF-IDF, Word2Vec and Doc2Vec are implemented to transform the texts into numeric representations. As TF-IDF and Doc2Vec create document embodiments, logistic regression can be used for classification. On the other hand, Word2Vec creates word embeddings, so that a LSTM architecture is applied to process the sentences as sequential input. BERT is downloaded as a pre-trained model and then fine-tuned on the manifestos, including the classifier in its architecture. The different models are compared and the best one is used for further experiments across time and country. The Python code of the analysis can be found on GitLab[1]. Google Colab is used to run the deep learning models as the available GPUs speed up training time tremendously.



Training data	Feature Engineering	Classifier
Raw and all three pre-processed source data sets	TF-IDF	Logistic Regression
Basic cleaned source data	Word2Vec	LSTM
Basic cleaned source data	Doc2Vec	Logistic Regression
Raw source data	BERT	

Figure 4.1.: Overview of different implemented models.

[1]https://gitlab.lrz.de/statistics/winter22/ba_sauter

4.1. Pre-processing

Three different pre-processing techniques are applied to the text data. The basic cleaning approach converts all words to lowercase and removes punctuation and stopwords. The latter are common words that do not contribute anything to the meaning of the text such as "a", "the" or "and" (Goyal et al., 2018). More advanced methods are stemming, converting all words to root words, and lemmatization (Khyani et al., 2021). The last technique additionally considers word meanings to identify the same word stems, also known as the word's lemma. For example, a lemmatizer recognizes that "better" can be derived from the word "good". Using the described methods, three different pre-processed data sets of the source and target corpus are created:

(i) Basic cleaning (lowercase; remove punctuation and stopwords)

(ii) Basic cleaning + Stemming

(iii) Basic cleaning + Lemmatization

All three data cleaning procedures are carried out by the Python library *Natural Language Toolkit (NLTK)*, using the pre-defined English stopwords and the methods "PorterStemmer" as well as "WordNetLemmatizer" (Bird et al., 2009). The histograms of word counts per text example for the raw and the pre-processed data sets (see Figure 4.2 and further details in Table A.2 in the appendix) show that the target data contains longer sentences (e.g. for the unprocessed data a maximum of 4306 and a median of 66 words) than the source data (e.g. for the unprocessed data a maximum of 140 and a median of 19 words). Moreover, the basic cleaning approach roughly halves the number of unique tokens (see Table A.1 in the appendix). Lemmatization and stemming further decrease the vocabulary size, but not the number of words per text example, so that the histograms of the differently pre-processed data overlap exactly.

The word clouds in Figure 4.3 give a descriptive overlook of important keywords for the topic "external relations" of the basic cleaned data (Mueller, 2018)[2]. A bigger font size indicates that this word is more frequent than others in a cluster of a defined maximum of 150 words. This illustration depicts similarities across the two data sets with common key words such as "government", "international", "people", "trade", "countries", "force" and "United Nations". An additional very important word in the target corpus seems

[2]https://github.com/amueller/word_cloud

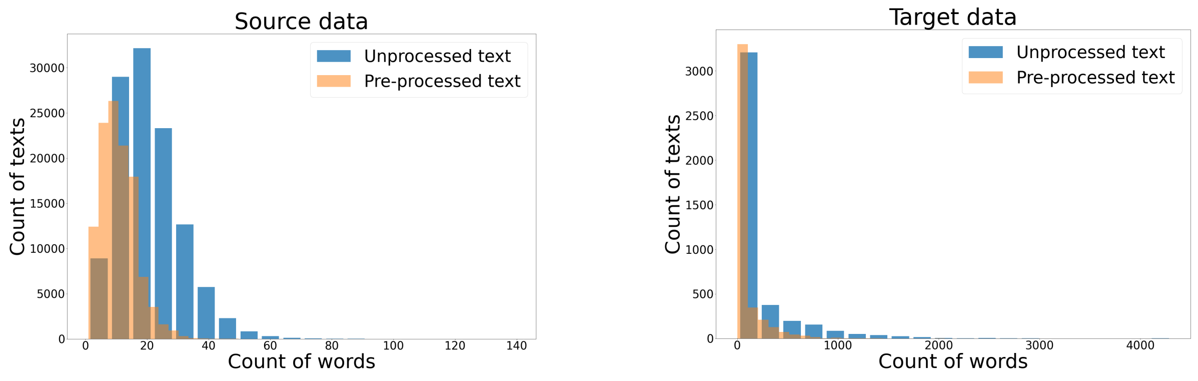


Figure 4.2.: Word count per text example of source (left) and target corpus (right) for the raw and pre-processed data. Note that the counts are the same for all three pre-processed data sets.

to be the country of origin "New Zealand". On the other hand, the word cloud of the source corpus contains several country names such as "Australia", "United States", "Africa, "UK", "EU", "Britain", "New Zealand" and "Ireland", indicating the different origins of the manifestos. In texts about external relations the own country name and the other mentioned key words are used often in both data sets. The word clouds of the other categories also result in interpretable clusters of key words, excluding the "no topic" word cloud which is a rather random accumulation of words (see appendix in Figure B.1 and B.2).

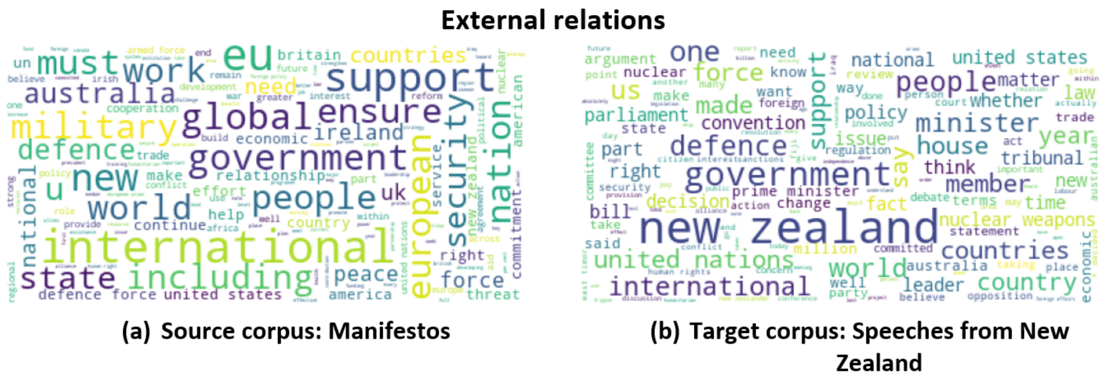


Figure 4.3.: Word clouds for the topic "external relations" of the source (a) and target corpus (b).

4.2. Training and evaluation of the different models

In the following the different models TF-IDF + LR, Word2Vec + LSTM, Doc2Vec + LR and BERT are trained on the source data. Therefore, the data is split into a train (80%, 92,328 observations) and test data set (20%, 23,082 observations). The latter is further divided into halves to create a test and validation set with 11,541 observations respectively for the deep learning models, namely BERT and the LSTM architecture. After splitting the data, the models are trained on the training data. In order to optimize the performance of each model, hyperparameter tuning is carried out based on the F1 score (macro), which considers the class imbalance best. Finally, the optimized models are used to predict the test and target data, so that the within-domain and cross-domain classification performance can be compared with regard to accuracy and F1 score (macro). On top of this, confusion matrices of each model of the source and target data can be found in the appendix (see Section C.3).

TF-IDF + LR

Fitting the TF-IDF + LR model on the source corpus is done similar to the paper by Osnabrügge et al. (2021), using the *scikit-learn* implementations "TfidfVectorizer", "GridSearchCV" and "LogisticRegression" with the same hyperparameters (Pedregosa et al., 2011). Firstly, feature engineering of the corpus is applied to transform the text data into a document-term matrix. Therefore, unigrams, bigrams and trigrams are created, dropping punctuation and capitalization. Then, the default English stopwords and the corpus-specific stopwords with a document frequency higher than 0.4 are removed. Rare words with an appearance in fewer than 10 documents are excluded as well. Finally, the TF-IDF value for each n-gram is calculated (see theory in Section 3.3.1), treating each entry of the training data as a document. This results in a document-term matrix, which is used as input for the multinomial LR. The hyperparameter C, which represents the inverse of the regularization strength, and the class weights are tuned via Grid Search in the hyperparameter space {C= [1,2,10,100], class weight= [None,'balanced']}. The balanced mode adjusts the weighting of the classes inversely proportional to the class frequencies, which is otherwise set to one (Pedregosa et al., 2011). The class weights are added as hyperparameter because the balanced mode might handle the class imbalance of the data better. The applied Grid Search uses 3-fold cross-validation, the Newton-CG optimization algorithm with L2-regularization as solver and the F1 (macro) as evaluation metric. After determining the best parameters, the model is tested on the left-out test set (within-domain classification) and the speeches (cross-domain classification). The test and target data are processed in the same way as the training data by the earlier fitted TF-IDF vectorizer to obtain the corresponding document-term matrices, which are then used as the input of the trained LR classifier. This procedure is conducted for the raw and all three pre-processed data sets (see Section 4.1) to see if lemmatization or stemming improve the model. However, rounding to the second decimal lead to almost the same results (± 0.01), indicating that the different pre-processing strategies result in the same performance outcomes. For this reason, the basic cleaned data set without any further pre-processing such as lemmatization or stemming is used for the further feature engineering of Word2Vec and Doc2Vec. Table 4.2 shows the evaluation of the final model on the basic cleaned test and target data with a document term-matrix of dimension (92328×15038) and the optimized hyperparameter C=2 and no class weights. Detailed confusion matrices can be found in the appendix (see Figure C.1). Comparing the evaluation scores to the results of the paper (Osnabrügge et al., 2021), the accuracy and F1 score (macro) are almost the same (see Table 4.2). Although different evaluation metrics are picked for Grid Search (paper: accuracy, here: F1 score (macro)), the resulting hyperparameters are the same. The minimal differences in the performance measurements can be explained by a different train-test-split with 75% in the paper and as the document-term matrix is fitted on the whole source corpus by Osnabrügge et al. (2021). This results in a bigger vocabulary, but slightly intersperses the separation of training and testing data.

	Own implementation		Original paper	
Metric	Test data	Target data	Test data	Target data
Accuracy	0.640	0.497	0.641	0.507
F1 score (macro)	0.525	0.436	0.523	0.450

Table 4.2.: Evaluation of TF-IDF + LR model on basic cleaned test and target data compared to results of the original paper (Osnabrügge et al., 2021).

Word2Vec + LSTM

In the following, word embeddings are created by Word2Vec and then used as input in a LSTM architecture to classify each text example. The general structure of the implemented neural network is illustrated in Figure 4.4 below.

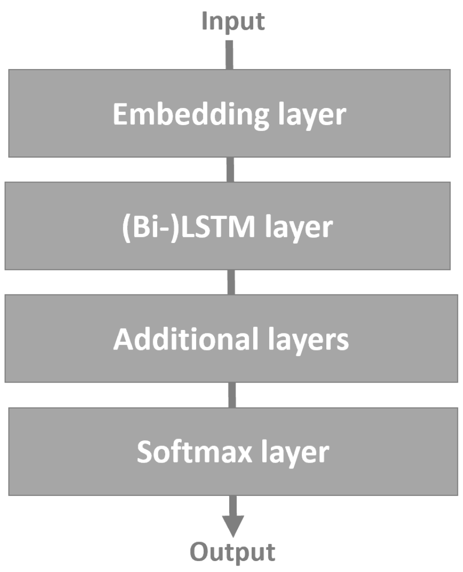
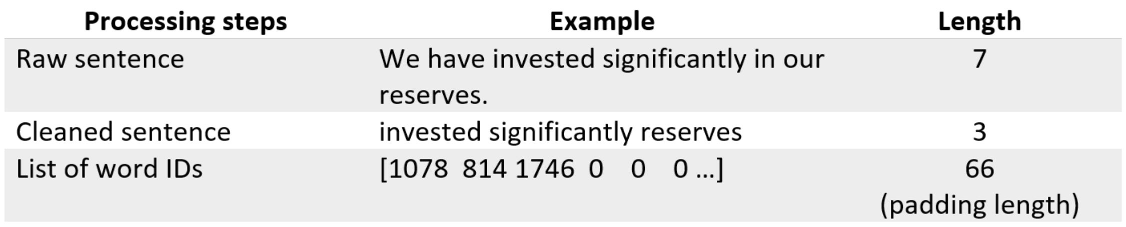


Figure 4.4.: General structure of the applied neural network of the model Word2Vec + LSTM.

First, the word representations are generated via the *gensim* implementation of Word2Vec on the basic cleaned training data set (Rehurek and Sojka, 2011). The created embeddings are looked up for every word of a sentence in the embedding layer, using an embedding matrix of the dimension (vocabulary size of $30,448$ $\times$ embedding size of 100). As the neural network expects a constant input length, short sentences need to be padded and long sentences need to be truncated to the same length. Therefore, the preprocessing text tokenizer of Tensorflow[3] (Abadi et al., 2015) is applied to the text input. The padding length is set to 66, which equals the maximum text example length of the basic cleaned data. The tokenizer also creates a dictionary, assigning every unique word to one word ID. As a result, each observation is processed to a sequence of word IDs of the same size (see sentence example in Figure 4.5). This representation is used as input of the LSTM. The embedding layer takes the input and the constructed word-embedding-lookup to match the words of the sentence and the corresponding embeddings. The output of the embedding layer is a matrix of size (padding length of 66 $\times$ embedding size of 100).

[3]https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer



Processing steps	Example	Length
Raw sentence	We have invested significantly in our reserves.	7
Cleaned sentence	invested significantly reserves	3
List of word IDs	[1078 814 1746 0 0 0 ...]	66 (padding length)

Figure 4.5.: Exemplary sentence processing where the last row represents the input for the LSTM.

Different layers can be stacked on the embedding layer. An exploratory analysis is carried out to find the most suitable architecture. The additional layers in Table 4.3 are investigated and evaluated based on the accuracy and F1 (macro) on the validation set. They are always followed by a last dense layer with eight neurons and a softmax activation function that returns the classification result. This procedure is repeated for CBOW and Skip-Gram, using the default embedding size of 100, categorical cross entropy as loss and the Adam Optimizer. To prevent overfitting, early stopping is implemented with a maximum of ten and a patience of three epochs while monitoring the validation loss. The Skip-gram method results in slightly better performance than CBOW (see Skip-Gram results in Table 4.3 and CBOW results in the appendix in Table C.1). The different layers show very similar scores with minimal better results of the two bi-directional LSTM layers followed by one additional dense layer. Using this architecture and extending the embedding size of the Skip-Gram model to 300 results in a F1 (macro) of 0.510 and does not meaningfully improve the performance of the model (see detailed results in Table C.2 in the appendix). The (Bi-)LSTM layers are always initialized with 128 units and the additional dense layer with 64 neurons.

After setting the architecture to "Embedding layer + Bi-LSTM + Bi-LSTM + Dense layer + Softmax" and the embedding hyperparameters to Skip-Gram with a vector size of 100, Grid Search is used to determine if a higher batch size or drop out can improve the model. The hyperparameter tuning of batch size= [32, 265] and dropout rate= [0.0, 0.2] of both Bi-LSTM layers lead to the optimal parameters of a batch size of 32 and a dropout rate of 0.2. No further hyperparameters of Word2Vec or of the neural network are changed and set to the defaults. After processing the test and target data in the same way as the training data, the final scores on both data sets are calculated (see Table 4.4). The detailed confusion matrices are shown in the appendix (see Table C.2).

Model	Loss	Accuracy	F1 (macro)
LSTM	1.095	0.615	0.492
Bi-LSTM	1.071	0.617	0.497
Bi-LSTM + Dense layer	1.073	0.621	0.498
Bi-LSTM + Bi-LSTM	1.063	0.626	0.503
Bi-LSTM + Bi-LSTM + Dense layer	1.053	0.629	0.507

Table 4.3.: Exploratory analysis of different architectures evaluated on the validation set. The word embeddings are created with Skip-gram and a vector size of 100. The first layer is always an embedding layer, followed by the additional layers described in the table and a last softmax layer for classification (see illustration 4.4).

Metric	Test data	Target data
Accuracy	0.632	0.492
F1 score (macro)	0.507	0.419

Table 4.4.: Evaluation of Word2Vec + LSTM model on basic cleaned test and target data.

Doc2Vec + LR

Doc2Vec is trained similarly to Word2Vec on the basic cleaned training data with the *gensim* package (Rehurek and Sojka, 2011). However, the model learns a document representation instead of word embeddings (see theory in Section 3.3.3). For this reason multinomial LR can be applied as classifier. Both methods of Doc2Vec, PV-DM and PV-DBOW, are fitted on the training data with a default vector size of 100. The resulting document embeddings are used as input for the LR, which is optimized by the Grid Search algorithm with the solver SAGA and a 2-fold cross-validation on the hyperparameter space C = [10, 100, 1000]. Compared to PV-DM with a F1 score (macro) of 0.412 and a optimized C of 1000, PV-DBOW works better with a resulting score of 0.564 and a C of 100. As a further experiment, the vector size of PV-DBOW is set to 300. However, this does not show any improvement with a resulting F1 score (macro) of 0.554 and a optimal C of 100. For the final fit, the PV-DBOW method is used with a vector size and C of 100. Because of earlier convergence problems of the solver, the maximal number of iterations is set to 4000 instead of the default of 100 in the final training procedure. Other possible hyperparameters of Doc2Vec or LR are set to their defaults. The test and target data is pre-processed by the fitted Doc2Vec model to create the document representations. The evaluation results of the trained LR on both data sets can be found in Table 4.5 and the confusion matrices in the appendix (see Table C.3).

Metric	Test data	Target data
Accuracy	0.588	0.398
F1 score (macro)	0.474	0.376

Table 4.5.: Evaluation of Doc2Vec + LR model on basic cleaned test and target data.

BERT

In comparison to the other models, BERT is not trained from scratch. For the implementation, the pre-trained models 'bert-base-uncased model'[4] and 'distilbert-base-uncased'[5] from Hugging Face are fine-tuned on the unprocessed training data set. The first model is the implementation of $BERT_{BASE}$, whereas the latter represents the $DistilBERT$ configuration (see theory in Section 3.4). The models are downloaded via the *transformer* package (Wolf et al., 2020). First, the data needs to be converted into token, segment and position embeddings by the downloaded tokenizer of the corresponding model. After that, the embeddings are used as input of the pre-trained model, that is then fine-tuned for five epochs on the training data set with the default "Trainer" implementation of *PyTorch* (Paszke et al., 2019). The best fine-tuned model is chosen based on the F1 score (macro) of the validation set. Finally, the model is used to make predictions on the test and target data set, which are also processed by the downloaded tokenizer in advance. The results in Table 4.7 show that $BERT_{BASE}$ is slightly better than $DistilBERT$. However, the latter takes about half an hour to train, whereas the training time of $BERT_{BASE}$ is about an hour with a T4-GPU provided by Google Colab.

	$BERT_{BASE}$		$DistilBERT$	
Metric	Test data	Target data	Test data	Target data
Accuracy	0.694	0.569	0.693	0.562
F1 score (macro)	0.595	0.512	0.584	0.501

Table 4.7.: Evaluation of fine-tuned $BERT_{BASE}$ and $DistilBERT$ on test and target data

[4]https://huggingface.co/bert-base-uncased
[5]https://huggingface.co/distilbert-base-uncased

4.3. Comparison of models

In the previous section, the four investigated models are trained on the training data and then evaluated on the test and target data. Table 4.8 summarizes the results of each final model ordered by its performance. Comparing the different approaches, BERT$_{BASE}$ clearly outscores the other methods, including the model of the original paper by Osnabrügge et al. (2021). BERT's performance is followed by the models TF-IDF + LR and Word2Vec + LSTM. Doc2Vec + LR results in the lowest scores. This is another success of the state-of-the-art NLP model BERT as the most suitable model for the within-domain as well as cross-domain classification of the manifestos and speeches. The implementation shows how transfer learning can be successfully used by taking an existing model such as BERT and fine-tuning it on a specific task. The model achieves an accuracy of 69% on the test set and 57% on the target data set, whereas the other models only reach scores between 59% -64% and 40%-50% respectively. The high performance is possible due to BERT's broader language understanding that the model learns during pre-training on the huge corpus. Another interesting finding is that the simple count-based TF-IDF approach works better than the more sophisticated methods Word2Vec and Doc2Vec. Overall, it is not surprising that the performance of the target corpus is lower than that of the source corpus. Thereby, higher scores of the source corpus lead to higher scores of the target corpus. All models learn patterns in the text data and are clearly superior to just guessing, which would result in an accuracy of 0.125.

Test set	BERT$_{BASE}$	TF-IDF + LR	Word2Vec + LSTM	Doc2Vec + LR
Accuracy	0.694	0.640	0.632	0.588
F1 score (macro)	0.595	0.525	0.507	0.474
Target set	**BERT$_{BASE}$**	**TF-IDF + LR**	**Word2Vec + LSTM**	**Doc2Vec + LR**
Accuracy	0.569	0.497	0.492	0.398
F1 score (macro)	0.512	0.436	0.419	0.376

Table 4.8.: Comparison of different models on test (top) and target data (bottom) ordered by decreasing performance.

Although BERT$_{BASE}$ achieves the best scores, *DistilBERT* is used in the following for further experiments across time and country. The performance is very similar (see Table 4.7), but the model takes only half the time to train. Before moving to the second research part, Figure 4.6 gives detailed information about the F1 score per category of *DistilBERT*. The evaluation shows that not all categories are equally well classified.

Whereas "welfare and quality of life", "external relations" and "economy" show F1 scores slightly above 70% in the source data, "political system" and "freedom and democracy" only reach scores below 60%. Looking at the evaluation of the target data per category, "welfare and quality of life" and "external relations" reach the highest F1 scores slightly above 60% and "freedom and democracy" only a score of 48%. Especially the "no topic" label is predicted incorrectly almost all the time for the source (4%) and target corpus (0%). This suggests that *DistilBERT* is not finding any patterns in the "no topic" category. Overall similar scoring patterns per category of the source and target data are found. This can be quantified by the Spearman correlation coefficient of 0.928, which calculates the rank correlation of the eight categories of both data sets. The confusion matrices of the target and source data show the classification per category in more detail and depict the same patterns (see Figure C.4 in the appendix). The other classifiers also demonstrate similar effects per class (see confusion matrices of the other models in Section C).

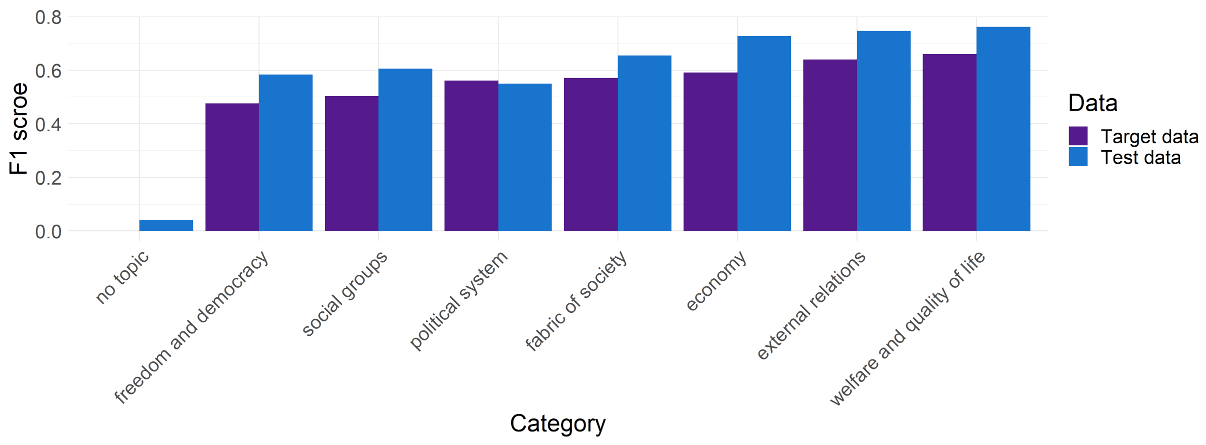


Figure 4.6.: F1 scores per category of *DistilBERT* for the test and the target data. Detailed information with category sizes are given in Table C.3 in the appendix.

4.4. Further experiments across time and countries

In this section *DistilBERT* is used for domain transfer across time and country. In order to investigate how the publishing year of the manifestos and the country of origin affects the model, the test-train-split is changed accordingly in the following analysis.

4.4.1. Test-train split across time

In the experiment across time the trained *DistilBERT* model from subsection 4.2 is used to make predictions in the future. As the training data only goes until 2018, the manifestos published after 2018 of the extracted corpus of 2021 are used as test set in the following. In Table 4.9 the earlier results of the test data from Table 4.7 (year $\leq$ 2018, 11,541 observations) and the results of the future data (year $>$ 2018, 20,970 observations) are depicted. The latter is predicted a little bit less accurate, but there is no big difference between the performance (accuracy: -0.018, F1 score (macro): -0.014). This suggests that the model can also be used to make predictions for manifestos in the future.

Metric	Test data	Future data
Accuracy	0.693	0.675
F1 score (macro)	0.584	0.570

Table 4.9.: Evaluation of *DistilBERT* on future data compared to results of previous test set.

4.4.2. Test-train split across countries

The differences between the countries of origin of the manifestos are investigated by splitting the extracted source data (version 2018) by country and use each split as test set. *DistilBERT* is trained seven times, excluding the data of one country each fit. In order to fine-tune the model, the training data is further randomly divided into a train and validation set with a 85%-15% split. The test set, including the text examples of only one country each time, is used to calculate the accuracy and F1 score (macro). The results of the evaluation metrics per country can be found in Figure 4.7 on the right side of each plot indicated as "Test split by country" on the x-axis. As comparison, the same data set is also randomly split with the usual 80%-10%-10% split for training, validation and testing. This model results in an overall accuracy of 0.691 and a F1 score (macro) of 0.579 on the test set. The scores are as expected almost equal to the results of *DistilBERT* on the source corpus of the paper (see Table 4.7). The evaluation metrics of this model are calculated per country and depicted in Figure 4.7 on the left side of each plot indicated as "Random test split evaluated per country" on the x-axis. Apart from the train-test split, nothing has been changed in the training procedure of the different models.

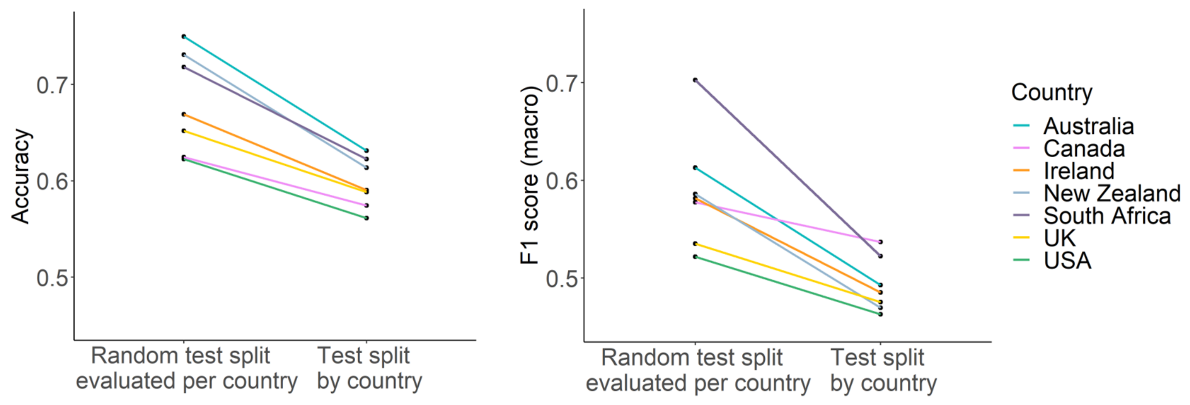


Figure 4.7.: Accuracy and F1 (macro) of *DistilBERT* for the random test split evaluated per country (result of one fitted model) and test split by country (result of seven fitted models).

Looking at the results of the experiment (see Figure 4.7), the random test split evaluated per country already indicates different scores per country. Comparing the two splitting procedures, the test splits per country show overall lower scores with an accuracy range from 0.56 (USA) to 0.63 (Australia) and a F1 score (macro) range from 0.46 (USA) to 0.52 (South Africa) than the random split with an accuracy range of 0.62 (USA) to 0.75 (Australia) and a F1 score (macro) range of 0.52 (USA) to 0.70 (South Africa). This indicates that the performance per country decreases if the model is not trained on text examples of the corresponding country. The finding suggests that there are differences in the manifestos between countries. This can be seen as transfer across countries.
In all cases, the USA result in the lowest scores. Australia achieves the best accuracy in both splitting procedures, whereas South Africa shows the best F1 score (macro). The detailed differences per country of the two splitting approaches can be found in Table 4.10. Canada shows the smallest difference, whereas Australia shows the highest with regard to accuracy. South Africa results in the highest difference looking at the F1 score (macro).
This outcome is influenced by the proportion of "no topic" observations per country, which are mostly misclassified (see Section 4.3). Canada in general as well as the test

split of South Africa do not include any example of this category (see Figure D.1), so that the F1 score (macro) is calculated on seven instead of eight categories, excluding the poorly predicted "no topic" category. South Africa contains exactly one misclassified "no topic" observation in the test split by country, explaining the striking decline of the F1 score (macro). The United States include the most "no topic" examples with 2% (see Figure D.1 in the appendix).

Difference	Canada	USA	UK	Ireland	South Africa	New Zealand	Australia
Accuracy	0.05	0.061	0.064	0.079	0.096	0.117	0.119
F1 (macro)	0.041	0.059	0.06	0.096	0.18	0.117	0.12

Table 4.10.: Difference in accuracy and the F1 score (macro) between the the two splitting procedures (see Figure 4.7). The values are ordered increasingly by the accuracy difference.

Furthermore, it is important to point out that the two different splitting procedures result in different sizes of the test sets. The random test split holds overall 11,452 examples (10% of the data), whereas the test split by country always contains all observations of one country. Taking the most common country New Zealand for example, 2,883 observations of this country are in the test set of the random split. On the other hand, the test split by country contains all 28,561 observations of New Zealand. This is visually depicted in Figure 4.8, comparing the sizes of the splits. The left plot shows the distribution of observations per country of the random test split, whereas the right plot depicts the seven test splits per country of the seven fitted models.

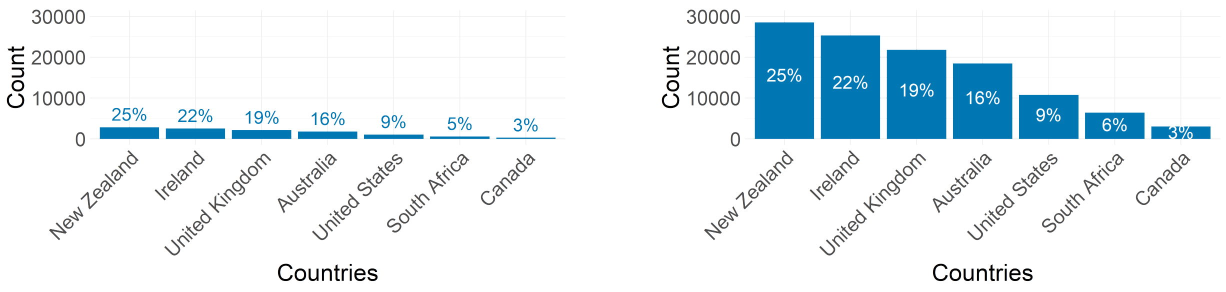


Figure 4.8.: Distribution of the countries of origin of the random test split (left) and the whole data set (right). The latter depicts the size of the test set for each test split per country.

5. Limitations and discussion

This thesis compares four different models to classify political texts in English into eight pre-defined categories. As languages are very different with regard to vocabulary and grammatical structure, the trained models cannot be applied to text data in another language. This also limits the countries of origin. Apart from South Africa, the chosen countries are all Western nations.

With regard to modelling, computational time and power are restricted within the framework of this thesis. Especially with regard to hyperparameter tuning, it is avoided to execute an exhausted search over a huge parameter grid. In further analysis, more time could be invested in hyperparameter optimization. For instance, the number of neurons of the Word2Vec + LSTM model could be tuned as well. In order to speed up the training process, GPUs provided by Google Colab are employed for the analysis. However, using these GPUs results in not fully deterministic implementations limiting reproducibility.

Having a closer look at the limitations of each model, TF-IDF is restricted to word frequencies and the vocabulary of the training data. The latter also holds for Word2Vec and Doc2Vec. The LSTM architecture of Word2Vec is also limited by the padding length of the text input. Especially as the target data consists of longer sentences than the source data, this restrain might lead to a reduced performance across domains. Furthermore, all three methods cannot handle ambiguity. BERT overcomes these limitations with dynamic padding, a huge pre-trained vocabulary and contextual embeddings. Although BERT is close to human language processing, there are still limitations. For instance, the model struggles with commonsense reasoning (Zellers et al., 2019; Vaswani et al., 2017) and does not have any political scientific knowledge like human coders.

Lastly, the models rely on the correct labeling by the Manifesto Project and the reliability of the coders. Mikhaylov et al. (2012) investigate this and point out some systemic problems of the coding scheme. The authors suggest that the coding scheme should be simplified to address reliability issues. As the broader eight topics are used in the analysis of this thesis instead of the detailed 54 categories, reliability should be ensured.

6. Conclusion

In the scope of this thesis, different NLP methods are used for pre-processing and subsequent classification of political texts. Four models are compared based on a machine learning (TF-IDF + LR), deep learning (Word2Vec + LSTM, Doc2Vec + LR) and transfer learning approach (BERT), where the latter one outperforms the others. Knowledge distillation demonstrates to be an effective method to speed up the training process while maintaining the performance of BERT. Moreover, cross-domain classification across time, country and modality seems to be a reasonable approach, saving time and effort by reusing an existing model. However, the decrease in accuracy compared to within-domain classification also shows some limits of preciseness of domain transfer. For instance, *DistilBERT* works better if text examples of the country of origin are part of the training data.

In future experiments, the "no topic" category should be handled differently. The results show that text examples of this class are almost always misclassified. This indicates that the models do not find any text patterns within this category. A possible idea would be to exclude this class and train a classifier on the seven remaining categories. Subsequently, all examples with a lower probability than some threshold could be classified to the "no topics". Another idea to improve the performance is to combine the four models into one by using ensemble methods (Ren et al., 2016). Moreover, a multi-language model could be used to account for texts in other languages. The model can be also further expanded to the 44 topics of the paper (Osnabrügge et al., 2021) or the 54 categories of the Manifesto Coding Scheme. Lastly, apart from using supervised learning, other methods such as the unsupervised topic modeling approach LDA (Blei et al., 2003) could be applied and compared to the results of this thesis.

A. Pre-processing details

Data	Vocabulary of source corpus	Vocabulary of target corpus
Raw text	72,333	40,188
Basic cleaning	33,280	22,539
Lemmatization	29,838	20,252
Stemming	22,706	15,513

Table A.1.: Unique tokens as vocabulary of each pre-processed data set compared to raw text.

	Source data		Target data	
Text cleaning	Raw	Pre-processed	Raw	Pre-processed
Maximum	140	66	4,306	2,062
Median	19	10	66	28

Table A.2.: Maximum and median of the word count per text example of the raw data and all three pre-processed data sets.

B. Word clouds

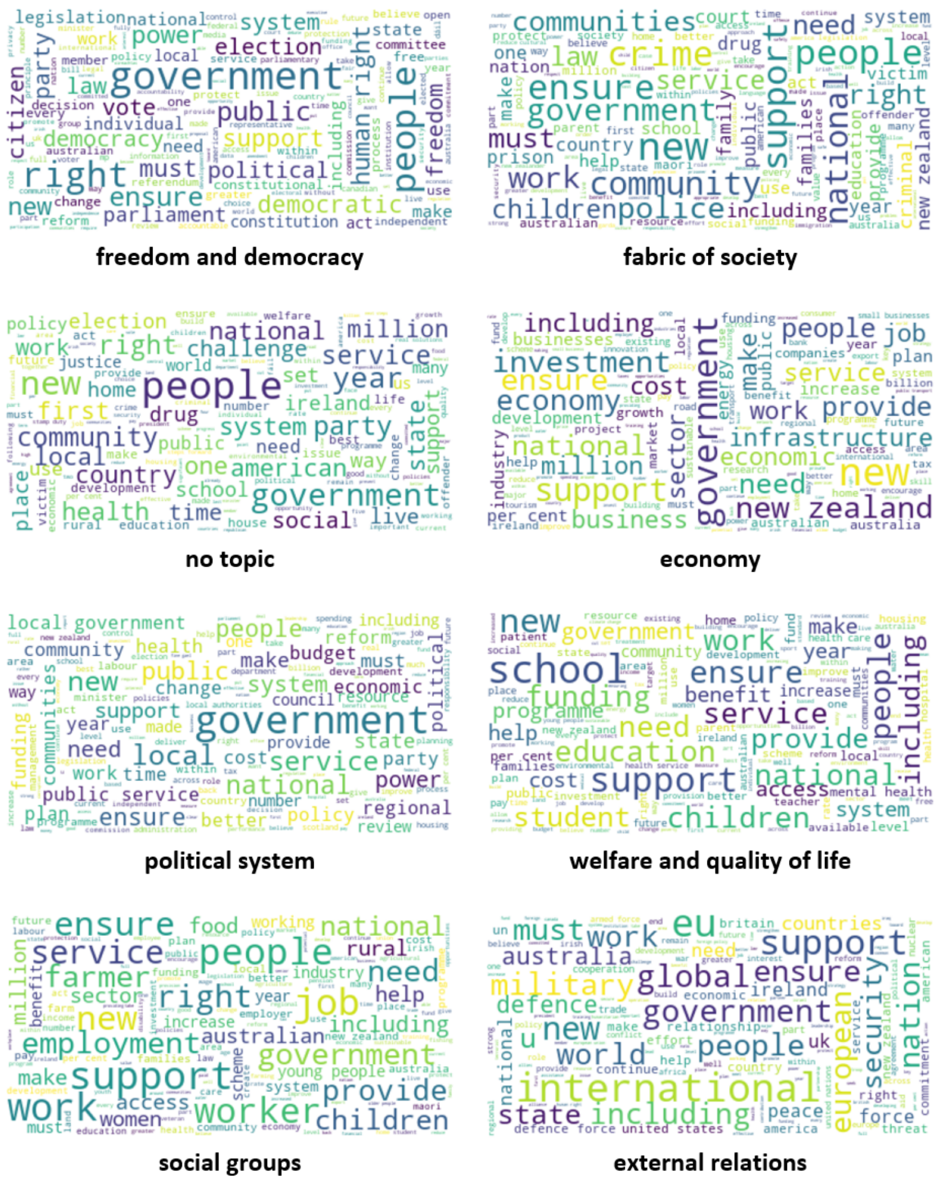


Figure B.1.: Word cloud per category of the source corpus.

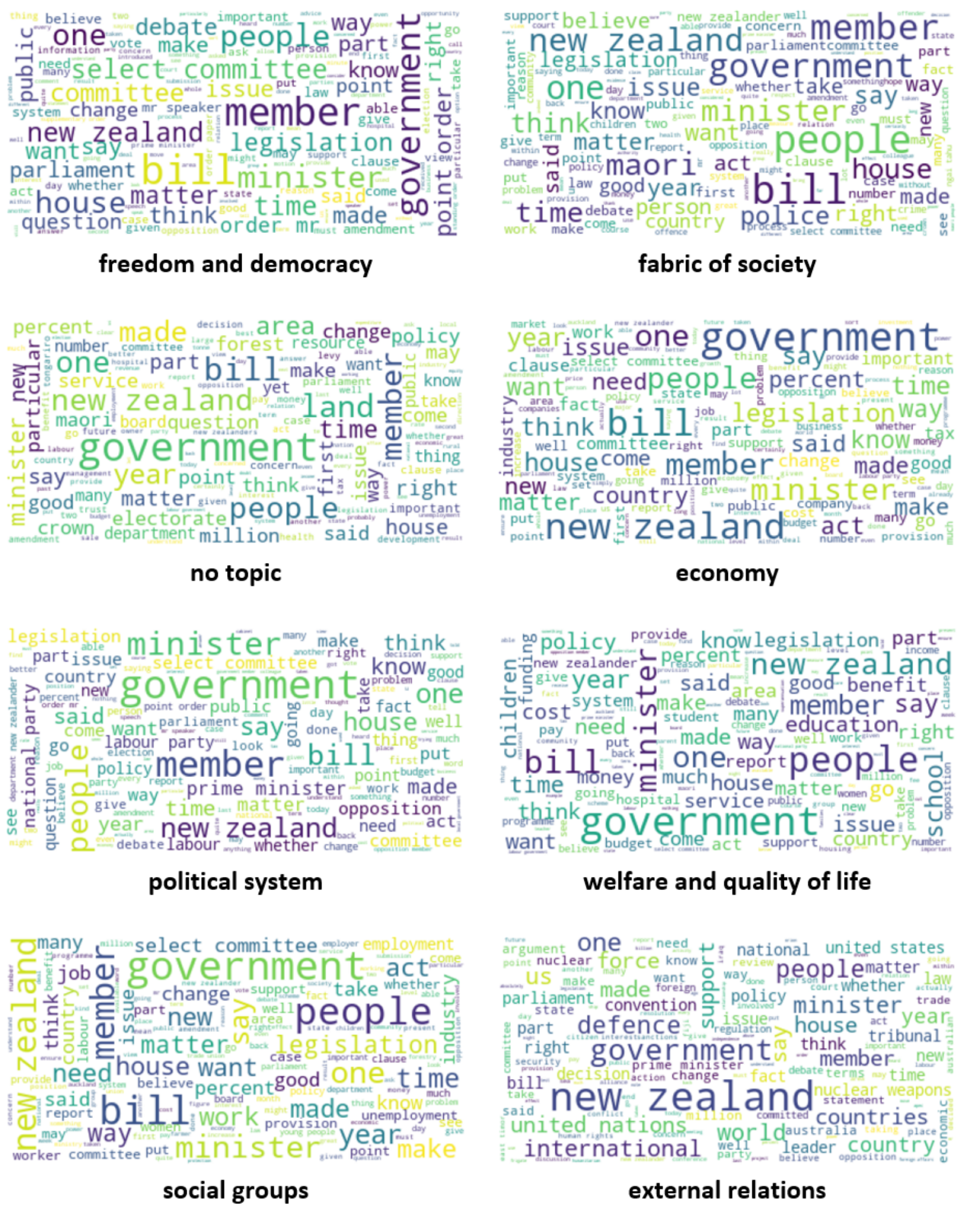


Figure B.2.: Word cloud per category of the target corpus.

C. Further detailed evaluation results of models

C.1. Exploratory analysis of Word2Vec

Model	Loss	Accuracy	F1 (macro)
LSTM	1.197	0.582	0.450
Bi-LSTM	1.166	0.588	0.468
Bi-LSTM + Dense layer	1.171	0.585	0.461
Bi-LSTM + Bi-LSTM	1.135	0.600	0.478
Bi-LSTM + Bi-LSTM + Dense layer	1.141	0.599	0.479

Table C.1.: Exploratory analysis of different architecture on validation set. The word embeddings are created with CBOW and a vector size of 100.

Model	Loss	Accuracy	F1 (macro)
LSTM	1.202	0.588	0.440
Bi-LSTM	1.069	0.620	0.500
Bi-LSTM + Dense layer	1.068	0.623	0.502
Bi-LSTM + Bi-LSTM	1.053	0.628	0.508
Bi-LSTM + Bi-LSTM + Dense layer	1.054	0.628	0.510

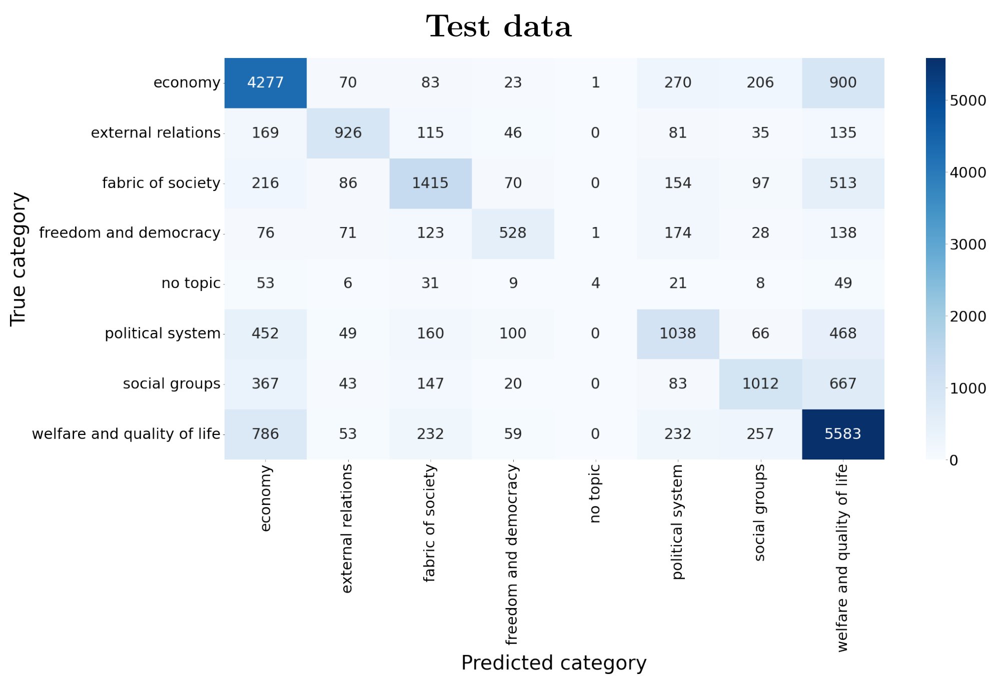
Table C.2.: Exploratory analysis of different architecture on validation set. The word embeddings are created with Skip Gram and a vector size of 300.

C.2. Evaluation of BERT per category

	Test data		Target data	
Categories	F1 score	support	F1 score	support
economy	0.728	2920	0.592	720
external relations	0.747	785	0.641	94
fabric of society	0.656	1245	0.572	432
freedom and democracy	0.584	564	0.477	545
no topic	0.041	91	0.000	192
political system	0.550	1179	0.561	1068
social groups	0.606	1149	0.504	325
welfare and quality of life	0.763	3608	0.660	789

Table C.3.: F1 scores per category for *DistilBERT* for test and target data.

C.3. Confusion matrices of models



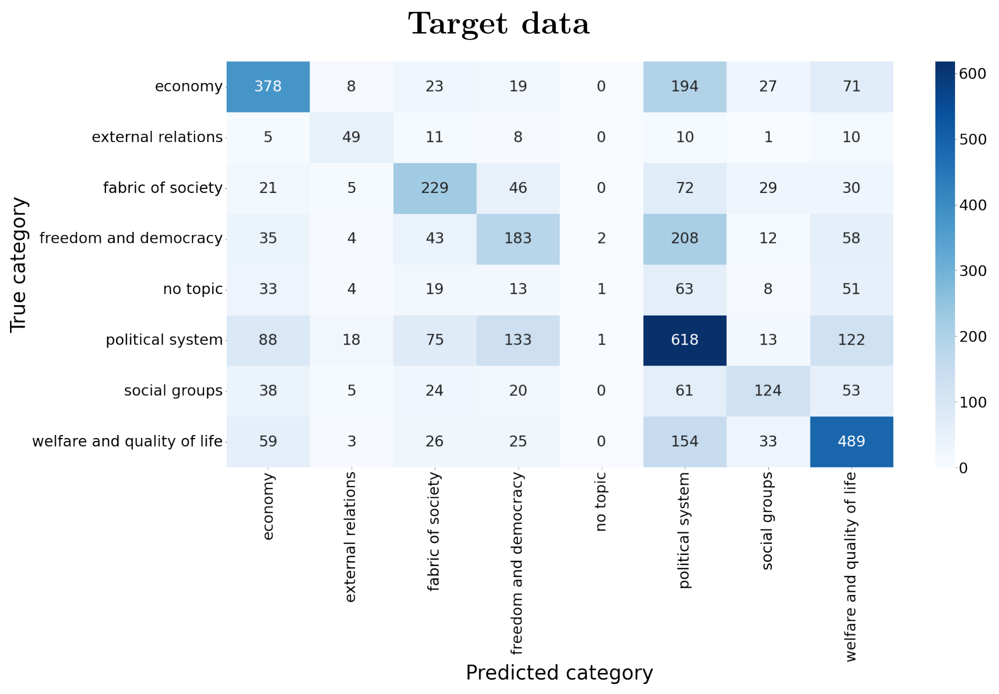
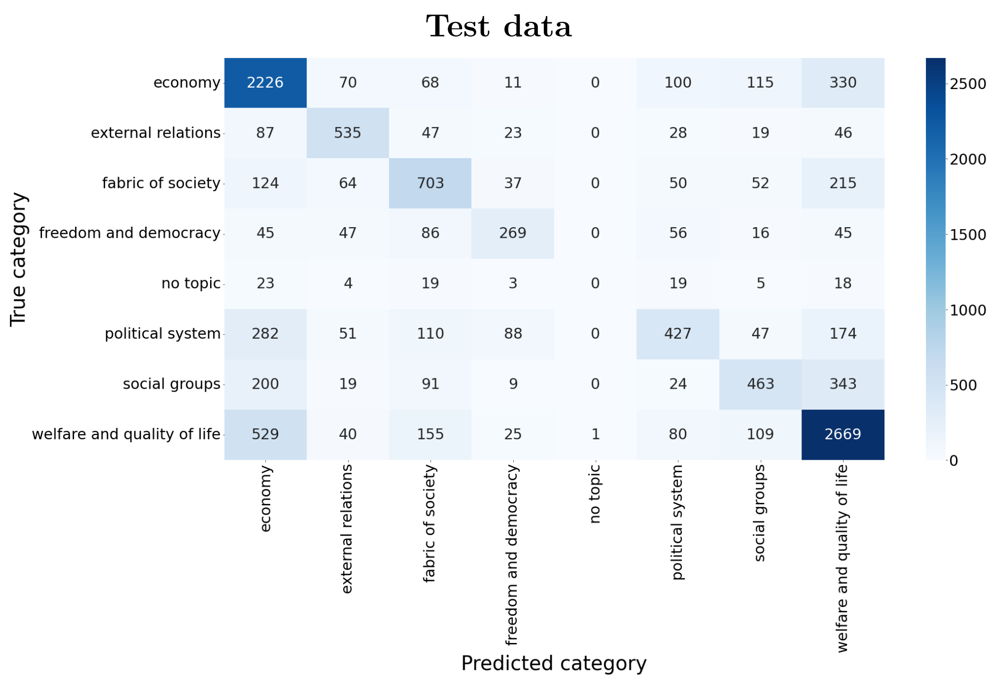
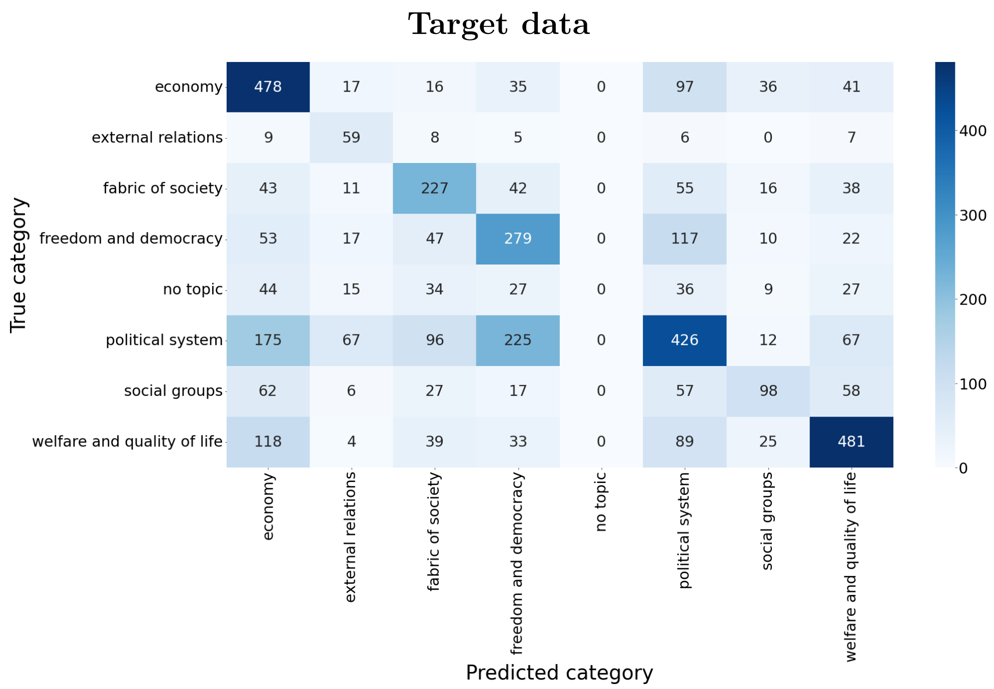


Figure C.1.: Confusion matrices of TF-IDF + LR model for test (above) and target data (below).

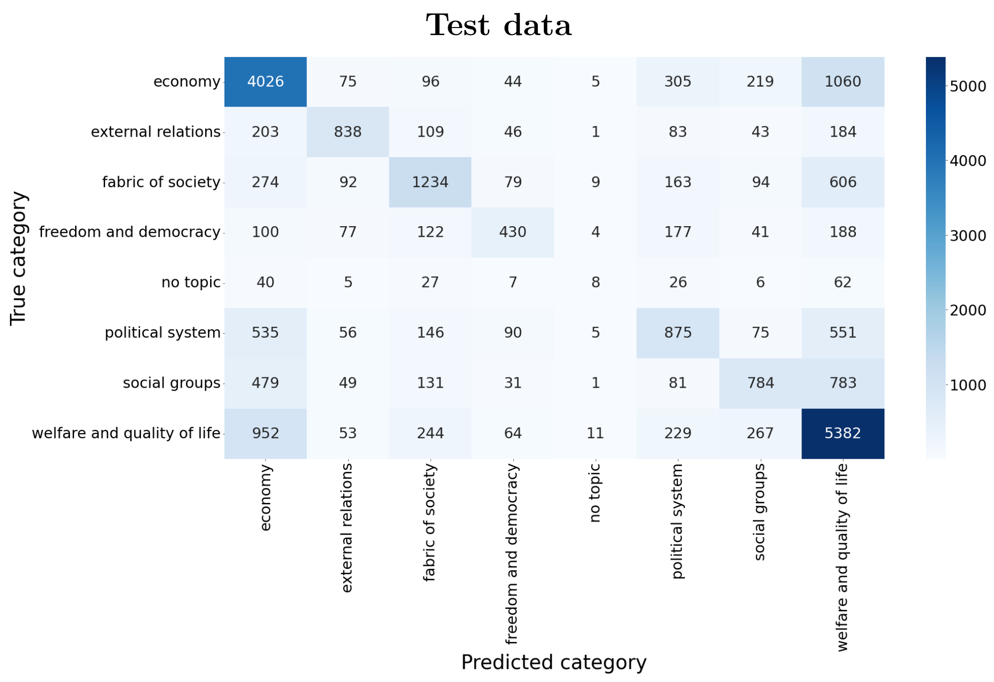


	economy	external relations	fabric of society	freedom and democracy	no topic	political system	social groups	welfare and quality of life
economy	2226	70	68	11	0	100	115	330
external relations	87	535	47	23	0	28	19	46
fabric of society	124	64	703	37	0	50	52	215
freedom and democracy	45	47	86	269	0	56	16	45
no topic	23	4	19	3	0	19	5	18
political system	282	51	110	88	0	427	47	174
social groups	200	19	91	9	0	24	463	343
welfare and quality of life	529	40	155	25	1	80	109	2669



	economy	external relations	fabric of society	freedom and democracy	no topic	political system	social groups	welfare and quality of life
economy	478	17	16	35	0	97	36	41
external relations	9	59	8	5	0	6	0	7
fabric of society	43	11	227	42	0	55	16	38
freedom and democracy	53	17	47	279	0	117	10	22
no topic	44	15	34	27	0	36	9	27
political system	175	67	96	225	0	426	12	67
social groups	62	6	27	17	0	57	98	58
welfare and quality of life	118	4	39	33	0	89	25	481

Figure C.2.: Confusion matrices of Word2Vec + LSTM model for test (above) and target data (below).



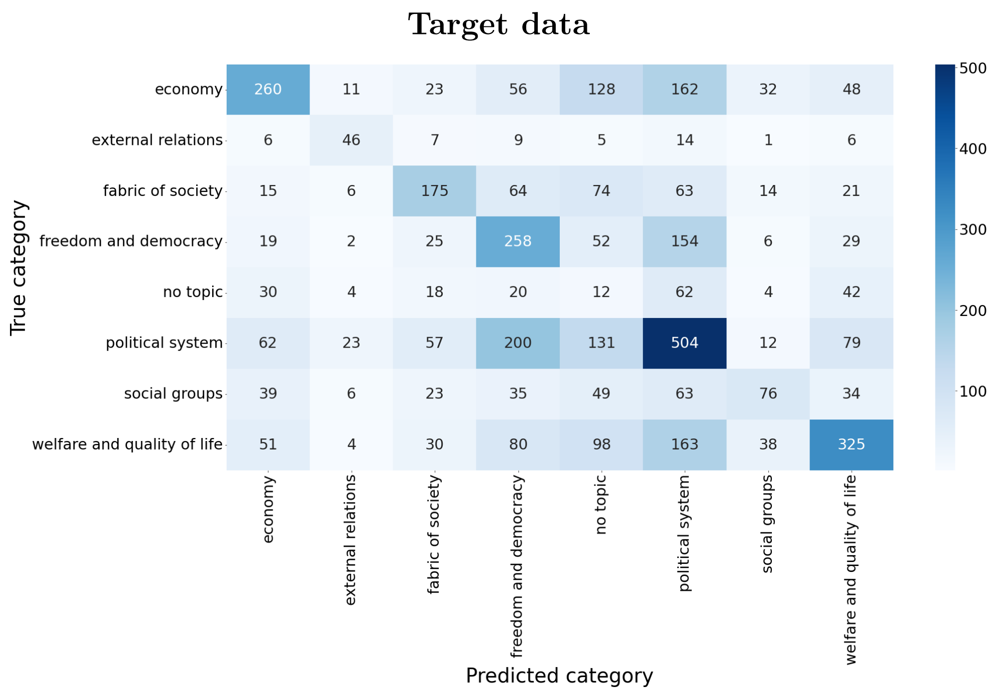
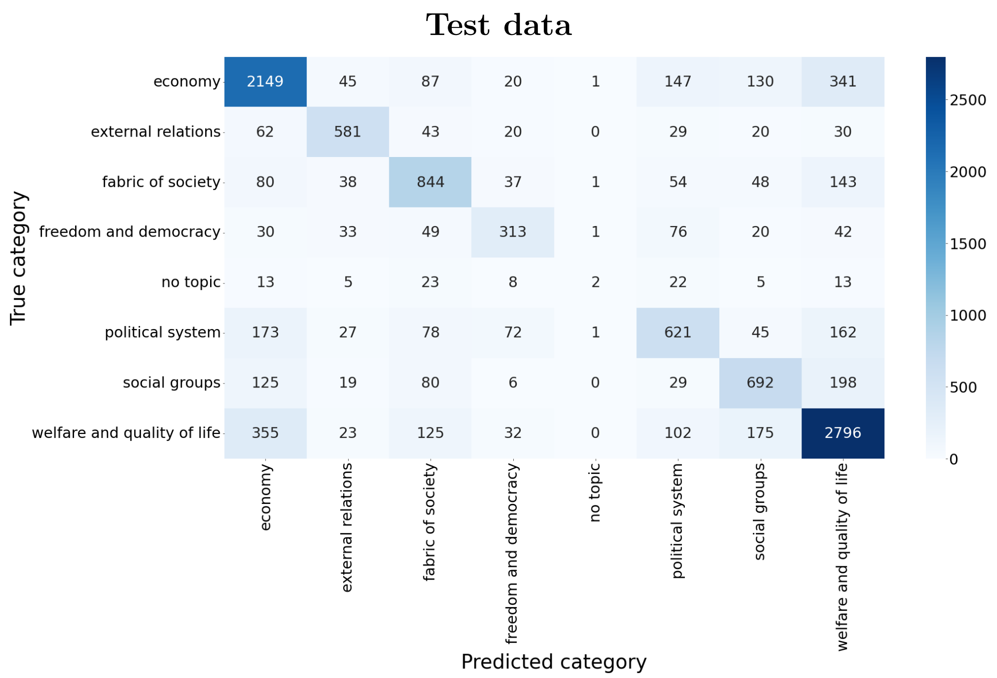
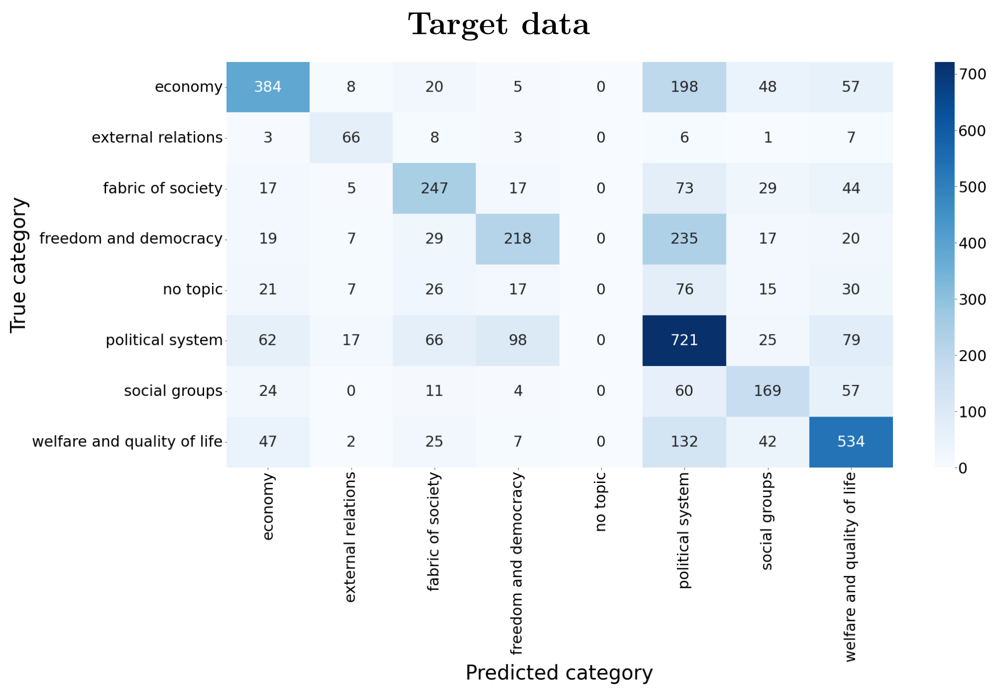


Figure C.3.: Confusion matrices of Doc2Vec + LSTM model for test (above) and target data (below).


Test data

True category \ Predicted category	economy	external relations	fabric of society	freedom and democracy	no topic	political system	social groups	welfare and quality of life
economy	2149	45	87	20	1	147	130	341
external relations	62	581	43	20	0	29	20	30
fabric of society	80	38	844	37	1	54	48	143
freedom and democracy	30	33	49	313	1	76	20	42
no topic	13	5	23	8	2	22	5	13
political system	173	27	78	72	1	621	45	162
social groups	125	19	80	6	0	29	692	198
welfare and quality of life	355	23	125	32	0	102	175	2796


Target data

True category \ Predicted category	economy	external relations	fabric of society	freedom and democracy	no topic	political system	social groups	welfare and quality of life
economy	384	8	20	5	0	198	48	57
external relations	3	66	8	3	0	6	1	7
fabric of society	17	5	247	17	0	73	29	44
freedom and democracy	19	7	29	218	0	235	17	20
no topic	21	7	26	17	0	76	15	30
political system	62	17	66	98	0	721	25	79
social groups	24	0	11	4	0	60	169	57
welfare and quality of life	47	2	25	7	0	132	42	534

Figure C.4.: Confusion matrices of *DistilBERT* for test (above) and target data (below).

D. Further analysis per country of origin

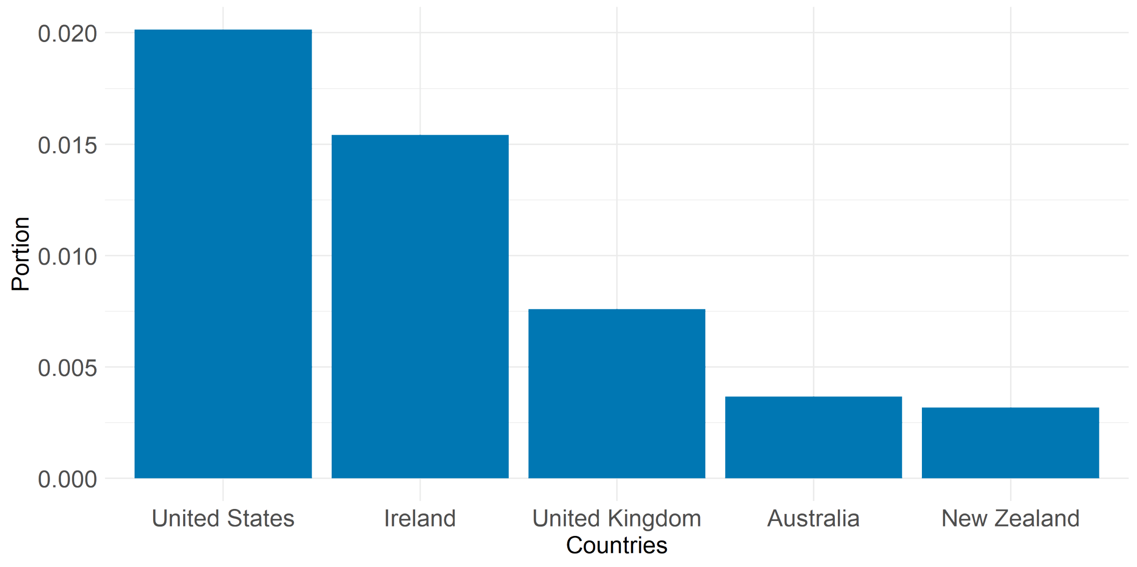


Figure D.1.: Distribution of "no topic" category per country of origin of the extracted source data (version 2018). Canada and South Africa are not depicted as Canada does not have any "no topic" examples and South Africa has only one.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Andonie, R. (2019). Hyperparameter optimization in learning systems. *Journal of Membrane Computing*, 1(4):279–291.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Böhmelt, T., Ezrow, L., Lehrer, R., and Ward, H. (2016). Party policy diffusion. *American Political Science Review*, 110(2):397–410.

Buciluă, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541.

Burscher, B., Vliegenthart, R., and De Vreese, C. H. (2015). Using supervised machine learning to code policy issues: Can classifiers generalize across contexts? *The ANNALS of the American Academy of Political and Social Science*, 659(1):122–131.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. (2011). Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398.

Defazio, A., Bach, F. R., and Lacoste-Julien, S. (2014). SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. *CoRR*, abs/1407.0202.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dobson, A. J. and Barnett, A. G. (2018). *An introduction to generalized linear models*. Chapman and Hall/CRC.

Esposito, M., Masala, G. L., Minutolo, A., and Pota, M. (2022). Natural language processing: Emerging neural approaches and applications.

Fahrmeir, L., Kneib, T., Lang, S., and Marx, B. D. (2021). Regression models. In *Regression*, pages 23–84. Springer.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*.

Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis lectures on human language technologies*, 10(1):1–309.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Goyal, P., Pandey, S., and Jain, K. (2018). Deep learning for natural language processing. *New York: Apress*.

Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of machine learning research*, 13(2).

Harris, Z. et al. (1954). Distributional hypothesis. *Word World*, 10(23):146–162.

Hinton, G., Vinyals, O., Dean, J., et al. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Janda, K., Harmel, R., Edens, C., and Goff, P. (1995). Changes in party identity: Evidence from party manifestos. *Party Politics*, 1(2):171–196.

Jurafsky, D. and Martin, J. H. (2022). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

Karabiber, F., Lewis, R., and Martin, B. (2022). Tf-idf - term frequency-inverse document frequency. Accessed: 2022-09-18.

Khyani, D., Siddhartha, B., Niveditha, N., and Divya, B. (2021). An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*.

Krause, W., Lehmann, P., Lewandowski, J., Matthieß, T., Merz, N., Regel, S., and Werner, A. (2018). Manifesto corpus. *Version:2018-2. Berlin: WZB Berlin Social Science Center*.

Larsen, J. and Hansen, L. (1994). Generalization performance of regularized neural network models. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 42–51.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

Lehmann, P. (2022). Manifesto project. Accessed: 2022-10-01.

Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*.

Liddy, E. D. (2001). Natural language processing.

Mikhaylov, S., Laver, M., and Benoit, K. R. (2012). Coder reliability and misclassification in the human coding of party manifestos. *Political Analysis*, 20(1):78–91.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *International workshop on artificial intelligence and statistics*, pages 246–252. PMLR.

Mueller, A. (2018). Wordcloud package. Accessed: 2022-10-05.

Olah, C. (2015). Accessed: 2022-09-18.

Osnabrügge, M., Ash, E., and Morelli, M. (2021). Cross-domain topic classification for political texts. *Political Analysis.*

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pilehvar, M. T. and Camacho-Collados, J. (2020). Embeddings in natural language processing: Theory and advances in vector representations of meaning. *Synthesis Lectures on Human Language Technologies*, 13(4):1–175.

Rehurek, R. and Sojka, P. (2011). Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2).

Ren, Y., Zhang, L., and Suganthan, P. (2016). Ensemble classification and regression-recent developments, applications and future directions [review article]. *IEEE Computational Intelligence Magazine*, 11(1):41–53.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Royer, C. W., O'Neill, M., and Wright, S. J. (2020). A newton-cg algorithm with complexity guarantees for smooth unconstrained optimization. *Mathematical Programming*, 180(1):451–488.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

Solomon, C. and Breckon, T. (2011). *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*. John Wiley & Sons.

Suiter, J. and Farrell, D. M. (2011). *The Parties' Manifestos*, pages 29–46. Palgrave Macmillan UK, London.

Tavits, M. and Letki, N. (2009). When left is right: Party ideology and policy in post-communist europe. *American Political Science Review*, 103(4):555–569.

Tsebelis, G. (1999). Veto players and law production in parliamentary democracies: An empirical analysis. *American political science review*, 93(3):591–608.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Wang, Y., Miller, D., and Clarke, R. (2008). Approaches to working in high-dimensional data spaces: gene expression microarrays. *British journal of cancer*, 98(6):1023–1028.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Xiao, C. and Sun, J. (2021). *Introduction to Deep Learning for Healthcare*. Springer Nature.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

Declaration of Authenticity

The work contained in this thesis is original and has not been previously submitted for examination which has led to the award of a degree.

To the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made.

Nadja Sauter