Natural Language Processing Project 1.2: Bias Detection

This project explores diverse models for bias detection in natural language processing (NLP) and text classification using a dataset of German news articles. The goal is to categorize articles into bias types (left, center, right). The models, varying in complexity and size, aim to extract insights from the data and are comprehensively evaluated for their effectiveness in this task.

1 INTRODUCTION

In this section, we will outline our overall approach for the project, detailing its structure and the sequential steps undertaken to achieve our objectives, which will be elaborated upon in this report.

Project's Structure. We've organized the project structure into the following files and directories.

- **datasets** The directory containing the German news dataset in CSV format.
- **logs** The directory holds all the logs related to the training, evaluation of various models, as well as the insights derived.
- **plots** The directory that houses all the plots extracting various insights from the data, encompassing both the training of different models and their results.
- **analyze.py** This script analyzes the data to extract various insights before any training occurs, saving them as plots.
- **apply.py** This script applies all defined models, trains them, tests their performance, and extracts insights from the evaluation, saving them as logs and plots.
- **dataset.py** This script contains a class that encapsulates the data and is responsible for implementing various pipelines for maniuplating the data such as pre- processing and vectorization.
- **models.py** This script defines and outlines the models that we will utilize for this project.
- **requirements.py** This script defines all the required packages to be able to run this project.
- **train.py** This script includes various classes responsible for training and testing a configuration of models on the provided dataset.

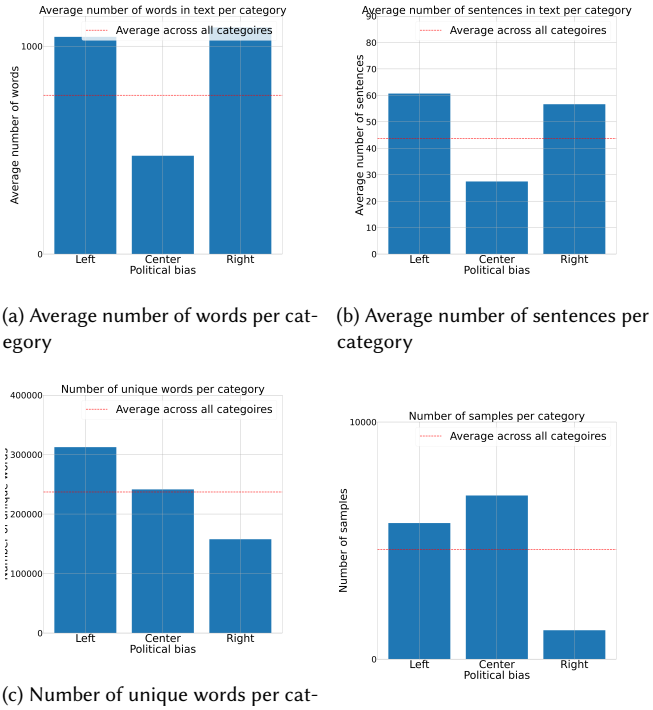
All the code is available in the GitHub repository here.

Project's Approach and Outline. In this project, we followed these steps to achieve the goal of extracting meaningful insights from the data and evaluating various models on it.

(1) Extracting general insights from the data
(2) implementing different pre-processing pipelines
(3) implementing different vectorization pipelines
(4) Training a naive Bayes model and evaluating it on the data, followed by a error analysis.
(5) Training a feed-forward neural network and evaluating it on the data, followed by a error analysis.
(6) Estimating the most similar words For 10 random words using Pointwise Mutual Information (PMI)
(7) Fine-tuning and evaluating two pre-trained Transformer-based models on the data, followed by a comparative analysis.

.

2 EXTRACTING GENERAL INSIGHTS FROM THE DATA



(a) Average number of words per category

(b) Average number of sentences per category

(c) Number of unique words per category

(d) Number of Samples per category

Fig. 1. Various plots

In this section, we will discuss various general insights that can be extracted from the data and their implications for the next steps in this project.

Average Number of words. As shown in Figure 1a, articles with a center political bias have a significantly lower average word count. This could lead to implications such as underfitting for that label compared to others.

Average Number of sentences. As shown in Figure 1b, the average number of sentences in articles with a center bias is significantly smaller compared to other labels. This indicates that articles with a center political bias are considerably shorter, which could lead to underfitting for that particular label, as discussed previously.

Number of unique words. As shown in Figure 1c, articles with a right political bias have the least amount of unique words, despite having relatively more samples and longer articles, as evident from

the previous two paragraphs. This could indicate a tendency to repeat words more frequently in articles with this political bias, making the vocabulary less extensive compared to other biases.

Number of Samples per category. We can observe the number of samples per category in Figure 1d. It is evident that there is a significant discrepancy, particularly noticeable in the right political bias category. Therefore, in all our setups, we will employ a *random oversampler* to balance out all the classes.

Important Words in each Category. We computed the most significant words in each category using the TF-IDF (Term Frequency-Inverse Document Frequency) metric. By averaging the TF-IDF scores of each word across all articles, we determined their importance and plotted the results for each category, as shown in Figure 2. It is evident that the most important words across all categories are relatively similar, focusing on political topics such as party names and other political terms.

(a) Important words in Left political bias (b) Important words in Center political bias (c) Important words in Right political bias

Fig. 2. Most Important Words per category

t-SNE Visualization. Finally, we visualize articles in a vector space by training a word2vec model [Mikolov et al. 2013] to generate word embeddings. We compute the average of all words in an article to represent its embedding. Using t-SNE for dimensionality reduction, known for its ability to reveal clusters, we observe in Figure 3 that the right political bias exhibits the strongest clustering. This suggests that articles in this category employ similar words and semantic meanings, making them more distinct and separable compared to other categories, which are less linearly separable.

3 PRE-PROCESSING PIPELINES

We have implemented various preprocessing pipelines that manipulate text in different ways to facilitate model training. Our preprocessing focuses on the concatenated text of article titles and bodies. The following preprocessing pipelines were applied, and different combinations of these pipelines can be utilized based on specific setup requirements:

Lowercasing: Lowercasing converts all text to lowercase, ensuring consistent handling of words regardless of their original case.

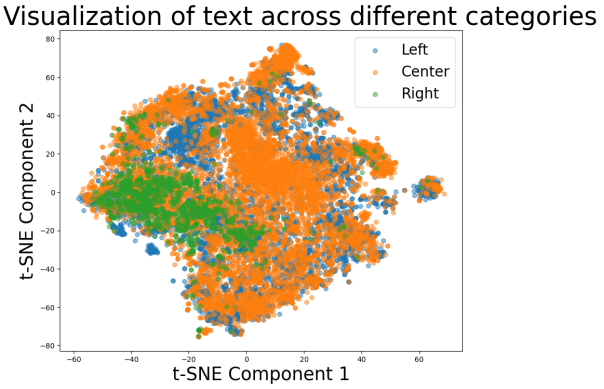


Fig. 3. t-SNE Visualization

Stop Words Removal: Stop words were removed using the German stop words list from NLTK, which filters out common words that do not contribute significantly to the meaning of the text.

Lemmatization: Lemmatization was performed using spaCy, which reduces words to their base or root form. This helps in standardizing words and reducing the dimensionality of the feature space.

Word Tokenization: Word tokenization was achieved using a tokenizer from NLTK, which splits text into individual words or tokens. This step is essential for further text processing and analysis.

4 VECTORIZATION PIPELINES

We have implemented various vectorization pipelines that transform the preprocessed text into vector representations (feature extraction). Each pipeline can only be applied individually. The ones we implemented include the following.

Count Vectorizer: Count Vectorizer converts a collection of text documents into a matrix of token counts. Each row represents a document, and each column represents a word in the vocabulary. The value in each cell denotes the frequency of a word in the document. The formula for the count of a term t in document d can be expressed as: $\text{Count}(t, d) = \text{Number of times term } t \text{ appears in document } d$

TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF transforms text into numerical vectors based on the importance of words in a document relative to a collection of documents. It scales down the impact of frequently occurring words across documents and emphasizes words that are unique to specific documents. The formula for TF-IDF of a term t in document d is:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

where TF (Term Frequency) measures how frequently a term occurs in a document, and IDF (Inverse Document Frequency) measures how important a term is across all documents in the corpus.

TF-IDF with Sublinear TF Scaling: This variant of TF-IDF applies a sublinear scaling to TF values before computing the final TF-IDF scores. It uses the formula: $\text{TF-IDF}_{\text{sublinear}}(t, d) = 1 + \log(\text{TF}(t, d)) \times \text{IDF}(t)$

Word Embeddings (Sentence Embeddings): Word Embeddings are generated by averaging the word embeddings produced by a trained Word2Vec model for all words in a sentence or document. This approach captures semantic meanings and relationships between words in a continuous vector space. The formula for averaging word embeddings in a sentence S with n words is: Word Embeddings$(S) = \frac{1}{n}\sum_{i=1}^{n}$ Word2Vec(w_i) where Word2Vec(w_i) represents the embedding vector of word w_i from the trained Word2Vec model.

5 EVALUATION METRICS

We are going to deploy the following evalutaion metrics to evaluate our models

Accuracy: Accuracy measures the proportion of correctly classified instances among all instances. It is calculated using the formula: $\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$

Precision: Precision measures the proportion of true positive predictions among all positive predictions made by the model. It is calculated as: $\text{Precision} = \frac{\text{True Positives}}{\text{True Positives+False Positives}}$

Recall (Sensitivity): Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions among all actual positive instances in the data. It is calculated using the formula: $\text{Recall} = \frac{\text{True Positives}}{\text{True Positives+False Negatives}}$

F1 Score: The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both measures. It is calculated as: $\text{F1 Score} = 2 \cdot \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision+Recall}}$

Macro Average. The *Macro Average* computes the average of a metric across all classes. Since we are dealing with three classes, this method calculates the mean value of each metric (such as precision, recall, and F1-score) across all classes to derive a single score. This approach was used to conclude the metrics' performance across the dataset.

6 NAIVE BAYES MODEL

A Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem with the naive assumption of independence among predictors.

The Naive Bayes classifier assumes independence among predictors $x_1, x_2, \ldots, x_n$:

$P(x \mid C_k) = P(x_1 \mid C_k) \cdot P(x_2 \mid C_k) \cdot \ldots \cdot P(x_n \mid C_k)$

To classify a new observation x, the Naive Bayes classifier selects the class C_k that maximizes the posterior probability $P(C_k \mid x)$:

$\hat{y} = \arg\max_k P(C_k \mid x)$

Training. We split the data into 80% training data and 20% test data, training the model using various preprocessing pipelines and vectorization methods. Given the exponential number of possible combinations, we conducted an ablation study due to resource limitations. This study aims to determine the effect of each preprocessing pipeline and vectorization method. We began with the fully preprocessed text, applying all pipelines, and then systematically removed one component at a time to observe its impact.

Evaluation. We evaluated the Naive Bayes model on 20% of the dataset using various vectorization methods and preprocessing pipelines. Table **??** presents the results. The count vectorizer showed the best performance across all metrics, likely benefiting from its simplicity, which mitigates overfitting risks. The probabilistic nature of Naive Bayes also benefits from stable probability estimates provided by the count vectorizer.

Interestingly, the Naive Bayes classifier demonstrated robustness to changes in preprocessing methods, maintaining consistent performance across different setups. However, removing stop words and lemmatizing the text appeared to have the most significant impact among the preprocessing pipelines tested.

Vectorization and Preprocessing	Precision	Recall	F1-Score	Accuracy
count vectorizer lower-stop words - lemmatize	0.834	0.836	0.794	0.833
count vectorizer stop words - lemmatize	0.838	0.838	0.834	0.836
count vectorizer lower - lemmatize	0.830	0.830	0.793	0.830
count vectorizer lower-stop words	0.836	0.836	0.794	0.836
count vectorizer lower-stop words - lemmatize - tokenizer	0.834	0.834	0.794	0.834
tfidf lower - stop words - lemmatize	0.733	0.733	0.747	0.733
tfidf stop words - lemmatize	0.734	0.734	0.745	0.734
tfidf lower - lemmatize	0.720	0.720	0.737	0.720
tfidf lower - stop words	0.745	0.745	0.751	0.745
tfidf lower - stop words - lemmatize - tokenizer	0.734	0.734	0.748	0.734
tfidf sublinear lower - stop words - lemmatize	0.711	0.711	0.722	0.711
tfidf sublinear stop words - lemmatize	0.710	0.710	0.720	0.710
tfidf sublinear lower -lemmatize	0.708	0.708	0.722	0.708
tfidf sublinear lower - stop words	0.715	0.715	0.723	0.715
tfidf-sublinear lower - stop words - lemmatize - tokenizer	0.710	0.710	0.721	0.710

Table 1. Performance Metrics for the Naive Bayes Model

Error Analysis.

Class	Precision	Recall	F1-Score	Support
Left	0.768	0.827	0.796	1372
Center	0.862	0.714	0.781	1411
Right	0.883	0.972	0.926	1356

The model performs best on the Right political bias, possibly because articles in this category are more similar compared to others, as inferred from the data analysis.

Since the model relies on word counts for prediction, errors may occur when words that typically appear in one class are found in another. For instance:

- **Article**: *AfD verurteilt Macrons Weltregierungserklärung*
  Predicted Label: Right, **True Label**: Left
  Explanation: Words like "AfD" and "verurteilt" are more common in Right articles in the training data.
- **Article**: *Merkel und Li wollen Zusammenarbeit bei Handel und Klima*
  Predicted Label: Left, **True Label**: Center
  Explanation Words like "Merkel" and "Zusammenarbeit" are more common in Left articles in the training data.

- **Article**: *Die CDU auf Linkskurs Erinnern Sie sich noch an die RoteSockenKampagne der CDU*
  Predicted Label: Left, **True Label**: Right
  Explanation Words associated with the Left political bias appeared in different contexts in right political bias articles.

7 FEED-FORWARD NEURAL NETWORK

In this section, we will explore the training of a feed-forward neural network on the dataset and evaluate its performance across various configurations.

Model's Configuration. The model architecture comprises three layers: an input layer, a hidden layer with a fixed size of 500 neurons using the ReLU activation function, and an output layer with three nodes corresponding to the three class labels. The output layer utilizes the softmax activation function to produce a probability distribution. For training, we employ the cross-entropy loss function.

Regularization. Given the propensity of neural networks to overfit, we will implement the following regularization methods:

(1) **Dimensionality Reduction** To reduce the dimensionality of the input vectors, which directly correlates with the number of parameters in the model, we will employ PCA (Principal Component Analysis). This is particularly beneficial for vectors with large dimensionality, such as those from count tfidf and sublinear tfidf vectorizer. PCA will reduce these vectors' dimensions from the vocabulary size of our dataset to a fixed input size of 200.
(2) **Dropout** In all our setups, we will use dropout regularization in front of the hidden layer with $p = 0.2$ to introduce variability in feature extraction and improve generalization.
(3) **Weight Decay** Another regularization method we utilized is weight decay with a value of 0.001, which mitigates overfitting by penalizing large weights and reducing variance.

Training. We trained the models using the ADAM optimizer for 50 epochs with a learning rate of 0.01 and a batch size. Instead of conducting a grid search over hyperparameters, we selected values that consistently led to a stable decrease in training loss, similar to successful setups. We are also using 80% of the data as training data and the other 20% as test data.

Evaluation. The results of the feed-forward neural networks evaluated across various setups can be found in Table **??**. In contrast to the Bayesian model, the TF-IDF sublinear setup demonstrates superior performance scores compared to all other configurations. This could be attributed to the neural networks' inherent flexibility and ability to extract features from complex patterns, unlike the Naive Bayes model. Additionally, TF-IDF sublinear performs better than the standard TF-IDF, possibly because TF-IDF sublinear produces more normalized values, leading to stable training and learning, in contrast to TF-IDF which may introduce high variance in input values. The preprocessing pipelines had minimal impact on the metrics; however, lowercasing and removing stop words appear to be the most effective configurations.

Error Analysis. We also conducted error analysis for the feed-forward neural network using the best performing setup, which includes TF-IDF sublinear vectorization with lowercasing and stop words removal as preprocessing steps.

Vectorization and Preprocessing	Precision	Recall	F1-Score	Accuracy
count vectorizer lower-stop words - lemmatize	0.675	0.825	0.743	0.772
count vectorizer stop words - lemmatize	0.789	0.634	0.703	0.834
count vectorizer lower - lemmatize	0.880	0.861	0.871	0.783
count vectorizer lower-stop words	0.825	0.826	0.826	0.850
count vectorizer lower-stop words - lemmatize - tokenizer	0.836	0.749	0.790	0.835
tfidf lower - stop words - lemmatize	0.738	0.854	0.792	0.812
tfidf stop words - lemmatize	0.730	0.862	0.790	0.815
tfidf lower - lemmatize	0.718	0.871	0.787	0.816
tfidf lower - stop words	0.732	0.892	0.804	0.827
tfidf lower - stop words - lemmatize - tokenizer	0.752	0.864	0.804	0.823
tfidf sublinear lower - stop words - lemmatize	0.756	0.907	0.824	0.840
tfidf sublinear stop words - lemmatize	0.766	0.898	0.827	0.842
tfidf sublinear lower - lemmatize	0.758	0.918	0.830	0.846
tfidf sublinear lower - stop words	0.754	0.923	0.830	0.850
tfidf sublinear lower - stop words - lemmatize - tokenizer	0.749	0.919	0.825	0.841
lower-stop words - lemmatize word embed	0.693	0.835	0.757	0.784

Table 2. Performance Metrics for the Feed-Forward Neural Network Model

	Precision	Recall	F1-Score	Support
0	0.754	0.923	0.830	1372
1	0.938	0.722	0.816	1411
2	0.898	0.910	0.904	1356

In comparison to the Naive Bayes model, our analysis shows that the neural network performed particularly well in identifying articles with a right political bias, surpassing the performance of other class labels. However, it also achieved superior metrics across other classes when compared to the best-performing Naive Bayes model.

We will analyze some samples where the model has made errors. These errors can occur because the model not only considers the frequency of words but also their contextual importance in different political biases.

- **Article**: "Matussek in Syrien: Matthias Matussek zieht in den Krieg"
  - **Predicted Label**: Right - **True Label**: Center
- **Article**: "Macron macht in Aachen den Trump: Der französische Präsident Emmanuel Macron und Bundeskanzlerin Angela Merkel demonstrieren"
  - **Predicted Label**: Left - **True Label**: Right
- **Article**: "Alice Weidel: Dieses Land wird von Idioten regiert (Video)"
  - **Predicted Label**: Right - **True Label**: Center

It can be inferred that certain words appearing in the error samples were deemed important according to TF-IDF in different political biases. This confusion highlights a weakness of such vectorization methods, namely their inability to grasp contextual nuances.

8 PMI BASED WORD SIMILARITY

In this section, we will determine the most similar words for 10 randomly selected words using the PMI (pointwise mutual information) metric. PMI measures the likelihood that two words co-occur more frequently than expected by chance alone. It is calculated using the following equation: $\text{PMI}(w_1, w_2) = \log\left(\frac{P(w_1 \cap w_2)}{P(w_1) \cdot P(w_2)}\right)$ where w_1 and w_2 are two words, $P(w_1 \cap w_2)$ is the probability of their co-occurrence, and $P(w_1)$ and $P(w_2)$ are their individual probabilities of occurrence.

PMI Configuration. Due to limited computational resources, we are not conducting the PMI measure across the entire vocabulary, which consists of more than 20,000 unique terms in our corpus. Therefore, we are restricting our analysis to the top 20,000 most important tokens determined by averaging the TF-IDF measure across all documents and filtering out less significant words. We also utilized a window size of 2 to determine words in similar contexts.

Evaluation. As observed in Table **??**, the most similar words estimated by PMI for 10 random words align logically with what one would expect to find in similar contexts across various articles. This demonstrates the effectiveness of PMI in estimating similarities between words.

Word	Most Similar Word
trump	donald
spd	deutschlandkoalition
merkel	angela
cdu	führungskrise
usa	militärmanöver
menschen	unantastbar
regierung	handlungsfähige
donald	urspräsident
schulz	spdkanzlerkandidat
türkei	eubeitrittsverhandlungen

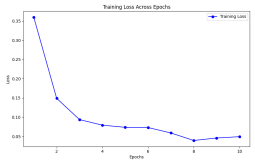
Table 3. Most Similar Words for Selected Words Using PMI

9 FINE-TUNING PRE-TRAINED TRANSFORMER-BASED MODELS

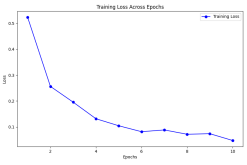
Transformer-based models excel in NLP tasks, surpassing other methods with their self-attention mechanism that understands text contextually rather than superficially. This robustness suits diverse NLP applications requiring text comprehension and generation. We fine-tune BERT [Devlin et al. 2019] and RoBERTa [Liu et al. 2019] on our dataset to evaluate their performance.

Model's Configuration. The model utilizes a transformer backbone with its specified tokenizer, which is the only pre-processing step needed for transformer-based models. The tokenized text is processed, and the pooled output (last hidden state of the [CLS] token) is extracted. We then add a hidden layer with 500 neurons, ReLU activation, and dropout, followed by an output layer with 3 neurons corresponding to the classes with softmax activation. Only the feed-forward layers are trained; the transformer layers are pre-trained on German text.

Training. We train the models using the Adam optimizer for 10 epochs with a batch size of 32 and a learning rate of 1e-4. This configuration helps to preserve the original pre-trained weights of the model and ensures stable training.



(a) Training Loss across Epochs for BERT



(b) Training Loss across Epochs for RoBERTa

Fig. 4. Training Loss across Epochs for the two transformer-based models

As seen in 4, the training loss across epochs for both transformer models decreases in a healthy manner, indicating successful training.

Evaluation. In this section, we will evaluate the two transformer-based models, comparing them with each other and with other setups.

Model	Precision	Recall	F1-Score	Accuracy
BERT	0.944	0.945	0.944	0.944
RoBERTa	0.851	0.843	0.838	0.841

Table 4. Performance Metrics for BERT and RoBERTa Models

As shown in the results 4, the transformer-based models performed exceptionally well on the data compared to other types of models. BERT significantly outperformed Roberta. This may be attributed to BERT's pre-training on the Next Sentence Prediction (NSP) task, which makes its pooler output more robust for a small amount of data. In contrast, Roberta generally achieves better results overall but requires more training data than BERT, which was not available in this case.

10 CONCLUSION

In this project, we applied various mechanisms to analyze the data and estimate the political bias of different articles, categorizing them as left, center, or right. We compared simpler methods, such as basic vectorization techniques and preprocessing steps, with more complex approaches like transformer-based models. This comparison illustrates the breadth of the NLP field and underscores the necessity of experimenting with different setups to achieve optimal performance.

REFERENCES

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]