Text Mining and Natural Language Processing

2024-2025

Tsuhuy Ecaterina, Brusati Lorenzo *Project:*

CAPITOL

Classifying American Political Ideologies Through Online Language

0. Introduction

The aim of this project is to develop and evaluate different models across the binary classification task of political tweets, specifically to distinguish between content aligned with either the Democratic or Republican party. The task involves analyzing a labeled dataset of social media tweets and designing a classification capable of identifying political affiliation based solely on tweet content.

Two main approaches to text representation were explored. The first involves TF-IDF vectorization, a sparse bag-of-words representation that captures term importance across the corpus. This representation was used as input to 2 traditional machine learning models, Logistic Regression and XGBoost, to establish strong baselines. The second approach leverages pretrained static word embeddings, specifically GloVe and a self-custom-trained FastText model, integrated into recurrent neural networks (RNNs)—notably LSTM architectures—to capture sequential and contextual patterns in the tweets.

The overall goal is to compare different combinations of preprocessing techniques and feature extraction methods to determine which pipeline yields the most effective performance on this political text classification task.

1. Data

The dataset used for this project is the *PoliticalTweets* dataset, made publicly available on the Hugging Face platform by the user "*Jacobvs*".

This dataset contains a large collection of tweets authored by United States politicians, and it is labeled according to the political affiliation of the author with a binary target: label 0 for Republican and label 1 for Democrat. Each tweet is stored along with metadata, including the text (both raw and cleaned), the name of the politician, and their political party.

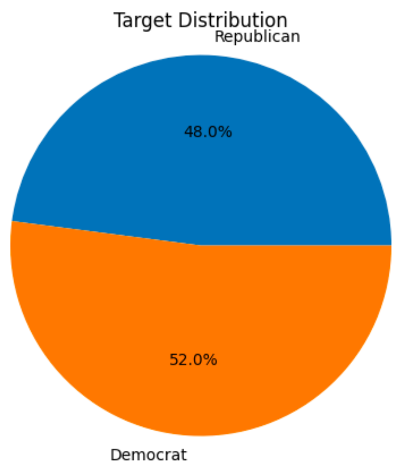
For the purposes of this research, we selected the first 30,000 entries from the training split due to computational costs of our virtual machine, while preserving the diversity and balance of the original dataset.

For this project, only the cleaned version of the tweet text and the label were used. Specifically, the column `text_cleaned` was employed as the input text data, and the column `label` as the binary classification target. This filtered dataset was exported to a CSV file and used throughout the project as the main data source.

To ensure consistency and reproducibility, the dataset was split using an 80/20 train-test split with stratification on the labels. No additional development set was created, as validation was performed during model training using internal splits (e.g., 20% validation split within the training data for neural models).

1.1 Class distribution

An important aspect of this dataset is its near-perfect balance between the two classes. Among the 30,000 initially selected tweets, 14,894 were authored by Democrats and 14,779 by Republicans. This results in a class distribution of approximately 49.3% Republican and 50.7% Democrat, visualized clearly in a pie chart. Such a balance is ideal for binary classification tasks and, as it prevents the model from being biased toward the majority class.



1.2 Text length analysis

A detailed statistical analysis was conducted on the length of tweets, both in terms of characters, words, and sentences.

The dataset reveals that the average tweet length across all users is approximately 214.82 characters. When comparing political affiliations, tweets authored by Democrats tend to be longer, with a mean length of 223.99 characters, whereas Republican tweets have a slightly shorter average of 204.86 characters.

Although the difference in length is not extreme, it suggests slight variation in rhetorical style or content density between the two parties. Democrats, on average, use more words and characters to express their messages.

1.3 Preprocessing pipeline

The preprocessing phase was designed to clean and standardize the tweet content before feeding it into the models. First, all text was converted to lowercase in order to eliminate any case-related variability that could interfere with token matching. Hashtags were then processed using the *wordninja* library, which splits concatenated words typically found in hashtags into their constituent parts, making them more interpretable for the model. For instance, a hashtag like *#BuildBackBetter* would be transformed into “Build Back Better”.

To remove non-textual noise, all emojis were stripped from the tweets using the *emoji* Python package. This was followed by a series of regular expression operations that cleaned the text further: user mentions (such as “@username”), hyperlinks, newline characters, numeric tokens, and all non-ASCII characters were eliminated. Punctuation was also removed to ensure uniform token formatting.

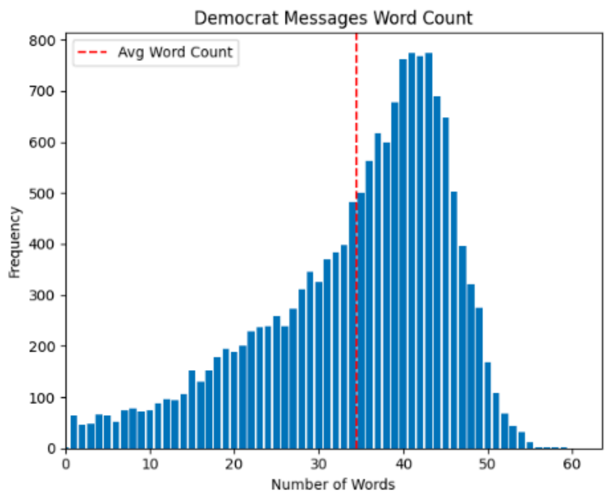
```python
def clean_text(text):

   text = str(text).lower()

   hashtags = re.findall(r"#\w+", text)

   for tag in hashtags:

       split_words = wordninja.split(tag.lstrip("#"))

       text = text.replace(tag, " ".join(split_words))

   text = emoji.replace_emoji(text, replace='')

   text = re.sub(r'\n', '', text)   # newline

   text = re.sub(r'@\w+', '', text)    # menzioni (@user)
   text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)  # link

   text = re.sub(r'\w*\d\w*', '', text)                                     # parole con numeri

   text = re.sub(r'[^\x00-\x7F]+', '', text)                                # caratteri non ASCII

   text = re.sub('[%s]' % re.escape(string.punctuation), '', text)          # punteggiatura

   return text
```

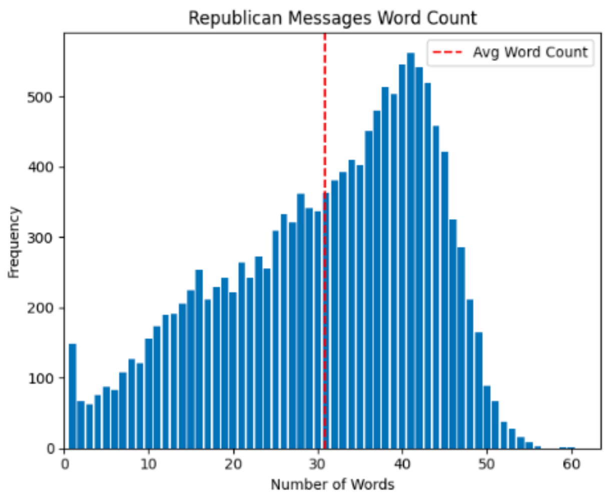
Furthermore, stopwords were removed using the standard list of English stopwords provided by the NLTK library. These are common words—such as “the,” “and,” or “to”—that typically carry little semantic value and tend to dilute the importance of meaningful features during training.

Finally, all tweets with fewer than two tokens after cleaning were discarded. This threshold was chosen to ensure that only linguistically meaningful and content-rich messages remained. As a result, 667 tweets were excluded.

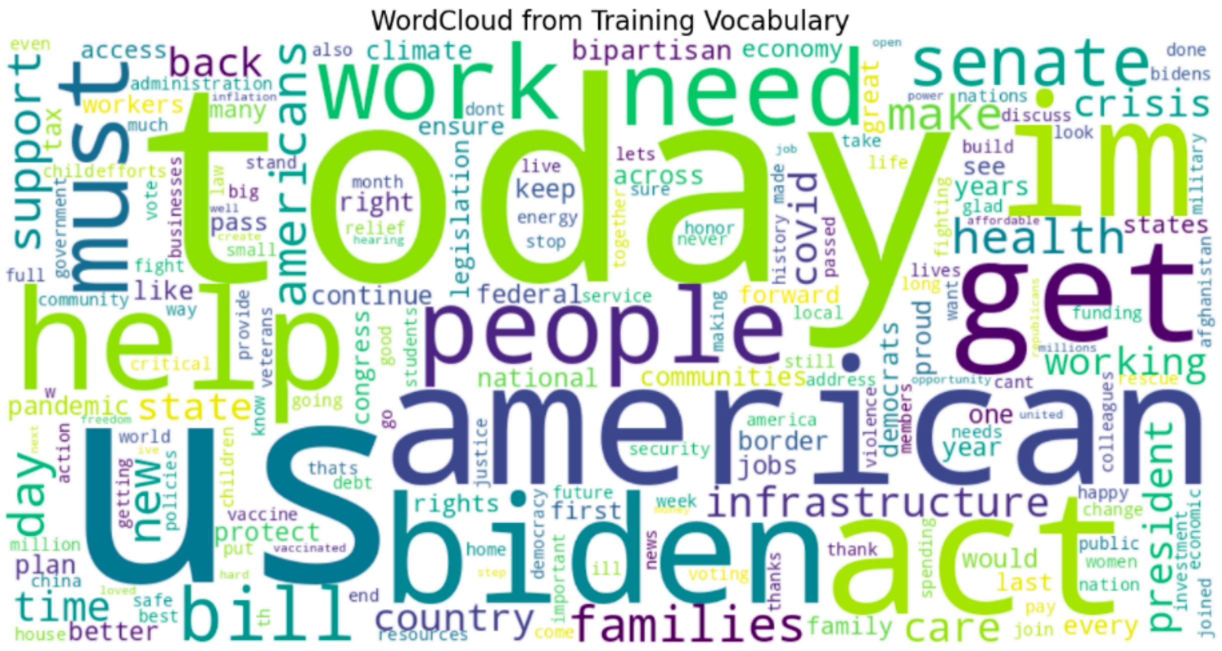
1.4 Sentence and word distributions

Using the NLTK tokenizer, we also measured the number of sentences per tweet. To better understand the distribution of tweet lengths, we created two bar charts showing the frequency of word counts for each political class. A red dashed line marks the average word count in each plot. Both distributions are skewed right, with a concentration around 25–40 words. A long tail extends to longer tweets: 432 tweets in the dataset exceed 50 words, which are considered statistical outliers in length. These were retained to preserve the full expressive range of the dataset.

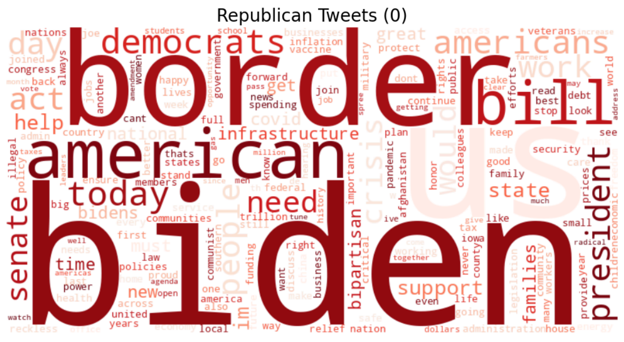


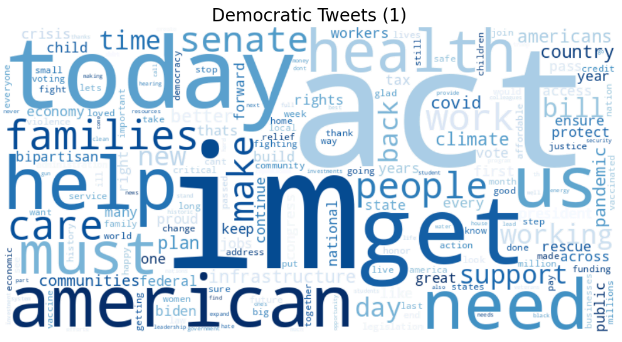


To complement the statistical and structural analyses of the dataset, a word cloud was generated to visually highlight the most frequently occurring terms across all tweets. This representation provides an immediate and interpretable overview of the dominant lexical items used by U.S. politicians on Twitter. The size of each word in the cloud is proportional to its frequency in the corpus, allowing for the identification of salient topics and rhetorical elements.



When produced separately for each political affiliation, the word cloud can also offer preliminary insight into the lexical and thematic distinctions between Democratic and Republican discourse.





2. Methodology

This section outlines the methodological framework adopted in the development, training, and evaluation of the various classification models explored in this project. It describes the strategies used for feature representation, the architecture and configuration of each model, and the rationale behind their selection.

2.1 Feature representation

Two distinct approaches to text representation were employed to convert tweets into machine-readable input features: sparse vectorization using TF-IDF and dense vectorization using pretrained word embeddings.

The first method employed a TF-IDF (Term Frequency – Inverse Document Frequency) vectorizer, a sparse bag-of-words representation that quantifies the importance of terms within the corpus by penalizing common words and emphasizing informative ones. This representation disregards word order but captures salient lexical features across the dataset. It was used as input to traditional machine learning classifiers and serves as a strong lexical baseline.

The second approach made use of dense, pretrained word embeddings, specifically GloVe (Global Vectors for Word Representation) and FastText. These embeddings encode semantic relationships between words by mapping them into continuous vector spaces where proximity reflects contextual similarity. While GloVe provides fixed embeddings trained on a large external corpus, FastText allows for subword information, offering robustness against OOV (out-of-vocabulary) words and morphological variations. It's really important to mention that FastText embeddings were trained on the project-specific dataset to better reflect domain-specific usage.

2.2 Model architectures

To assess the impact of both feature types, each was coupled with a classification model appropriate to its nature. The TF-IDF vectors were fed into two traditional classifiers: Logistic Regression and XGBoost. Logistic Regression is a linear model well-suited for high-dimensional sparse data. XGBoost was selected for its strong performance in classification tasks and its ability to capture non-linear relationships in the data.

In contrast, the dense word embeddings were used as input to recurrent neural network (RNN) architectures, specifically Long Short-Term Memory (LSTM) networks. LSTMs are well-suited for sequential data and are capable of modeling long-range dependencies by maintaining hidden state across time steps. For each embedding type, an identical LSTM structure was used to highlight which of the two embedding models was better. Each LSTM was initialized with the corresponding pretrained embeddings and then the output was passed to a couple of dense layers with a sigmoid activation function to produce the binary classification output.

2.3 Training configuration and optimization

Traditional classifiers (Logistic Regression and XGBoost) were implemented using the scikit-learn and XGBoost libraries. The LSTM models were implemented using TensorFlow and Keras. Each model was trained for a maximum of 10 epochs. A 20% validation split was used within the training set. The binary cross-entropy loss function was minimized using the Adam optimizer with an initial learning rate of 0.01. Dropout layers were added after the embedding and LSTM layers to further regularize the network and mitigate overfitting.

2.4 Tokenization and padding

To prepare the tweet texts for input into neural network models, a structured tokenization and padding procedure was applied. This process ensures that all text inputs are standardized into a fixed length format suitable for training and inference within sequential architectures such as LSTMs.

The Keras *Tokenizer* was first instantiated with the vocabulary size derived from the training corpus. An out-of-vocabulary (OOV) token was also included to handle any unseen words during inference. This tokenizer was trained only on the training portion of the dataset to avoid data leakage, mapping each unique word to an integer index. Once the tokenizer was fitted, each tweet was transformed into a corresponding sequence of integers representing the tokenized word indices.

Subsequently, all sequences were padded to a uniform length to ensure that the input shape matched the expectations of the neural network layers. The maximum sequence length was determined based on the 99th percentile of the training set sequence lengths, which was 31 tokens. This threshold was chosen to preserve almost all tweets while reducing the impact of extreme outliers and avoiding unnecessary computational overhead from overly long inputs.

This procedure resulted in padded input tensors of shape (23243, 31) for training and (5811, 31) for testing.

2.5 Embedding initialization

To provide semantically rich input representations, two types of word embeddings were explored: GloVe and FastText. GloVe embeddings were loaded from the publicly available *glove.6B.100d.txt* model, which provides 100-dimensional vector representations trained on a large corpus of web and Wikipedia text. Each word in the model's vocabulary was mapped to a corresponding GloVe vector. Words not found in the GloVe vocabulary (OOV terms) were initialized as zero vectors. Out of 27,096 total tokens, 20,938 had a matching GloVe embedding, leaving approximately 6,157 as OOV words.

FastText embeddings were trained from scratch on the same PoliticalTweets dataset to ensure domain-specific representation. The FastText model was initialized using a skip-gram architecture (sg = 1) with a vector size of 100, a window size of 5, and a minimum word frequency (min_count) of 2 to focus on relevant terms. Training was performed for 10 epochs using 4 worker threads to parallelize the process. The model was saved to disk for reuse, avoiding retraining on subsequent runs.

After training, the FastText word vectors were extracted and used to construct an embedding matrix analogous to the GloVe setup, enabling their integration into the deep learning model.

We illustrate the difference between GloVe and FastText by examining how each encodes the word “trump”, highlighting the ability of fastext to capture not only the embedding but also similar forms like "trumps", "trumpera", or "trumpappointed".

2.6 LSTM architecture

The core architecture for both the GloVe-based and FastText-based models was a LSTM network.

The input was first passed through an Embedding layer, initialized with either the GloVe or FastText matrix, set to be non-trainable.

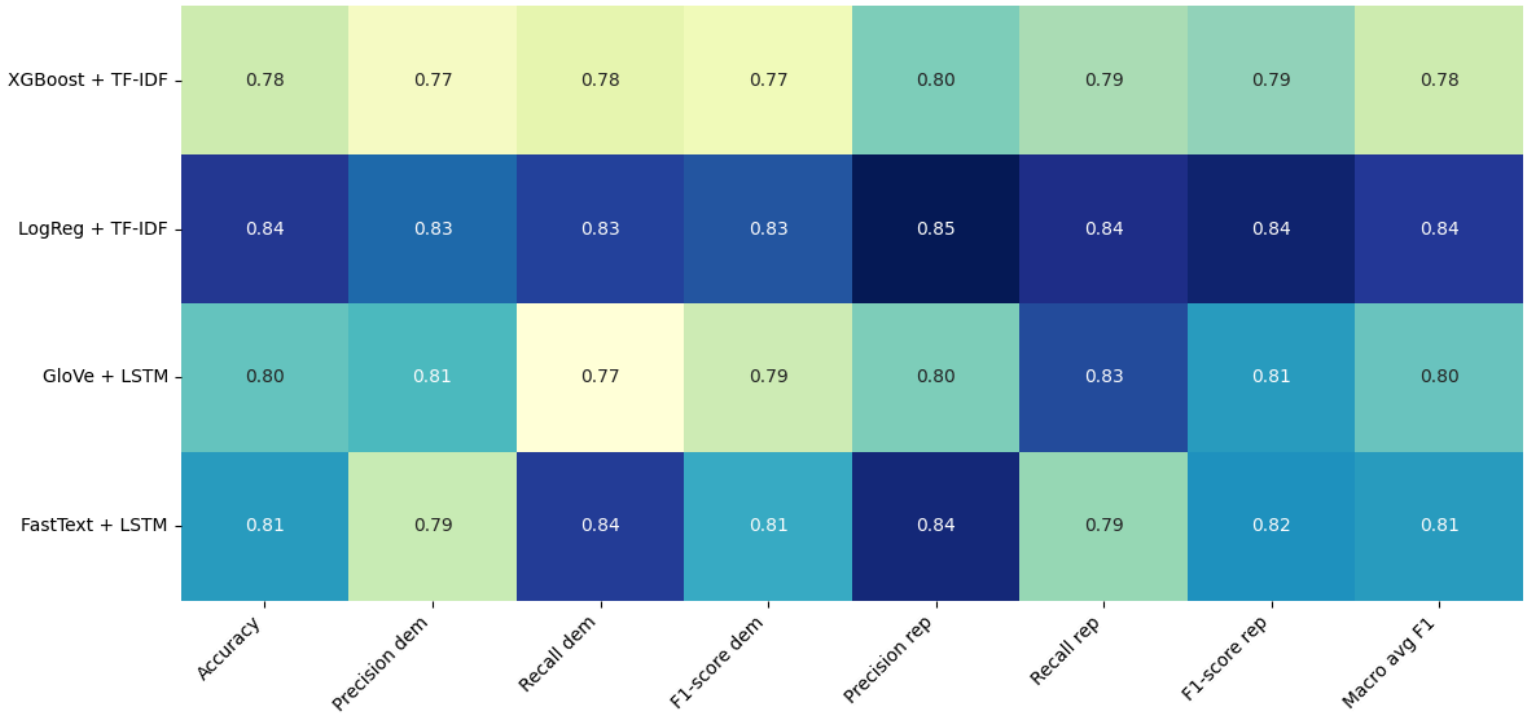
The embedded input was then processed by a unidirectional LSTM layer with 128 units. Subsequently, two fully connected Dense layers with ReLU activation were applied, each followed by a Dropout layer with a dropout rate of 0.2. Finally, a Dense output layer with a sigmoid activation function was used to generate a probability score for binary classification.

The models were compiled using the binary cross-entropy loss function and optimized using the Adam optimizer with an initial learning rate of 0.001. Each model was trained for up to 10 epochs with a 20% validation split on the training set to monitor generalization performance. No early stopping was applied.

3. Result and Analysis

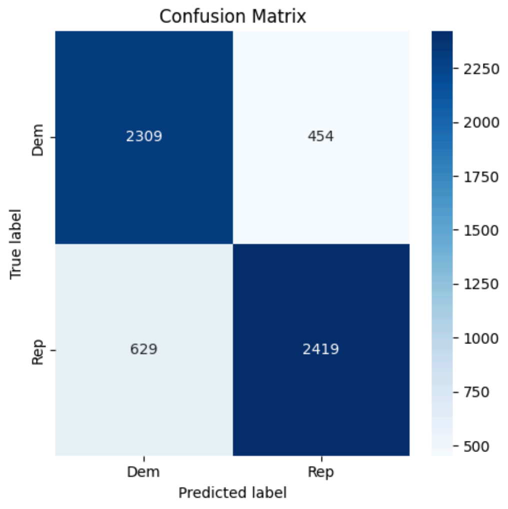
This section presents the empirical results obtained from each classification model, followed by a discussion of their performance metrics and an examination of representative outputs.

The evaluation was based on a held-out test set using standard classification metrics: accuracy, precision, recall, and F1-score. These metrics were computed for both classes in the binary task (left-leaning vs. right-leaning tweets). The results are summarized in the confusion matrices and performance comparison heatmap below, which report the classification effectiveness across all four models: XGBoost and Logistic Regression with TF-IDF features, and LSTM-based classifiers using GloVe and FastText embeddings.



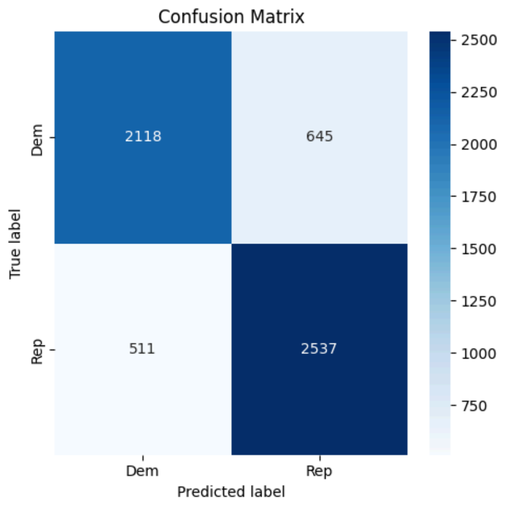
	Accuracy	Precision dem	Recall dem	F1-score dem	Precision rep	Recall rep	F1-score rep	Macro avg F1
XGBoost + TF-IDF	0.78	0.77	0.78	0.77	0.80	0.79	0.79	0.78
LogReg + TF-IDF	0.84	0.83	0.83	0.83	0.85	0.84	0.84	0.84
GloVe + LSTM	0.80	0.81	0.77	0.79	0.80	0.83	0.81	0.80
FastText + LSTM	0.81	0.79	0.84	0.81	0.84	0.79	0.82	0.81

The Logistic Regression model with TF-IDF features achieved the best overall results, with an accuracy of 0.84 and a macro-averaged F1-score of 0.84. It performed consistently well across both classes, with high precision and recall, particularly for the Republican class (precision 0.85, recall 0.84). This indicates that the model effectively captures the most discriminative terms for each class, benefiting from TF-IDF's ability to highlight distinctive vocabulary in political discourse.



Our self-trained FastText with LSTM also performed competitively, with an accuracy of 0.8202 and a macro-averaged F1-score of 0.8315. Its confusion matrix (shown on the left) reveals strong performance in classifying Democratic tweets (recall of 0.85), although slightly less effective on Republican tweets (recall of 0.79). The model's low test loss (0.4985) and minimal overfitting, as shown in the training plots, confirm its robust generalization capabilities.

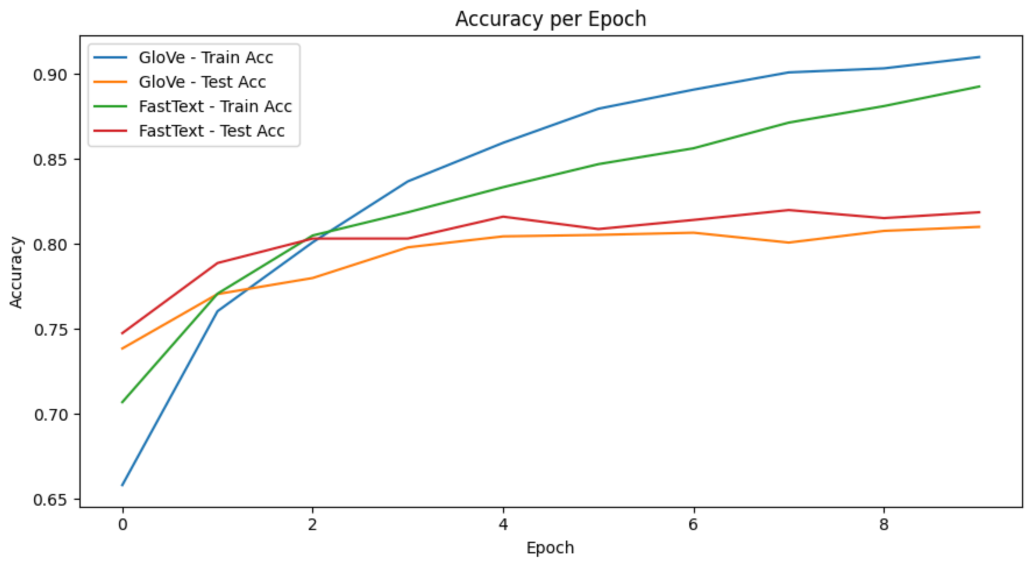
The GloVe with LSTM model achieved slightly lower results, with an accuracy of 0.7933 and a macro-averaged F1-score of 0.80. Notably, the recall for Democratic tweets (0.77) was lower than that of Republican tweets (0.82), suggesting a slight bias in favor of the Republican class. Despite good training convergence (final training loss around 0.26), the test loss plateaued (0.5631), indicating some generalization limitations.

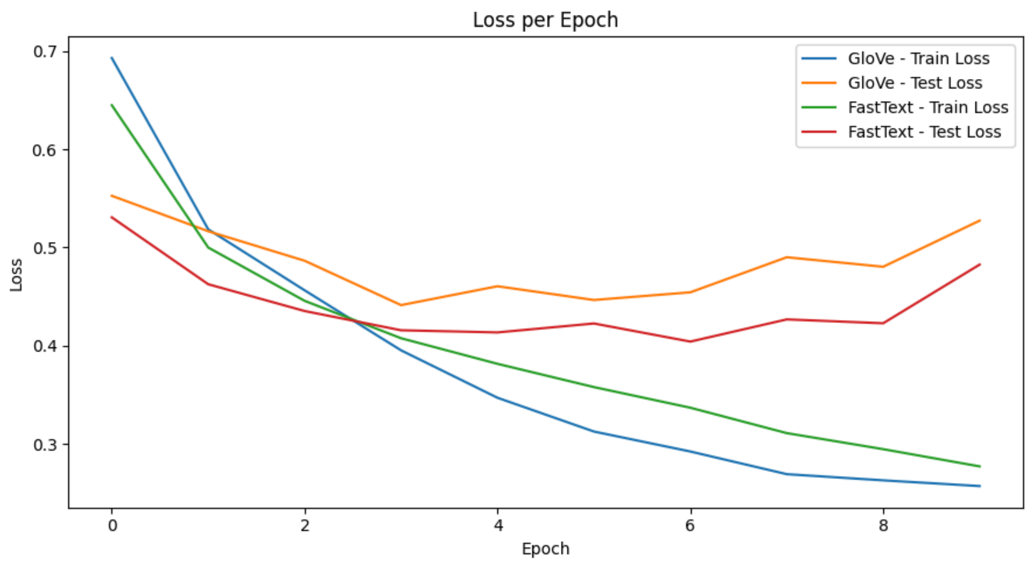


XGBoost with TF-IDF showed the lowest performance among all models, with an accuracy of 0.78 and a macro F1-score of 0.78. Its confusion matrix shows more misclassifications for both classes, particularly in the Democratic class (recall 0.79), suggesting that tree-based models might not be as well-suited to sparse high-dimensional representations in this specific task.

3.1 Training curves

The training curves of the LSTM models reveal additional insights. While both GloVe and FastText embeddings allowed the models to learn meaningful features, FastText showed a smoother convergence and less overfitting, likely due to its subword information capturing better features in political language.





These quantitative results are consistent with manual inspection of the outputs. For instance, tweets using colloquial, mispelled, or morphologically varied words (e.g., “libs”, “leftist”) were better captured by FastText, thanks to its subword-level embeddings. Conversely, TF-IDF-based models like Logistic Regression excelled when strong lexical cues were present, as these are heavily weighted by TF-IDF.

4. Conclusion

This project has been a valuable opportunity to deepen our interest in Natural Language Processing and to explore the challenges of political text classification. We particularly enjoyed experimenting with different representation techniques and model architectures, and we are especially proud of having trained a custom FastText model directly on our dataset. This allowed us not only to tailor embeddings to the political domain but also to "play" with semantic neighborhoods, exploring how politically charged terms relate to each other in vector space.

Although the final model performances are not state-of-the-art (but the state-of-the-art in our project we personally think is the acronym), the project has been extremely educational. It gave us practical insight into the entire NLP pipeline—from data preprocessing to embedding strategies and model evaluation—and helped us appreciate the complexity and creativity involved in designing NLP solutions from scratch. The hands-on experience, especially with domain-specific word embeddings, has been both technically enriching and genuinely enjoyable.

4.1 Issues encountered

The first major obstacle during project development was the limited performance of the Deepnote virtual machine. CPU usage was extremely inefficient, which forced us to reduce the dataset to only 30,000 rows to maintain feasible training and evaluation times.

Additionally, we faced compatibility issues between various libraries, which slowed down experimentation and required frequent environment adjustments.

Finally, the project involved downloading large pre-trained models such as GloVe, which resulted in significant disk usage and long setup times, further complicating development within a constrained cloud environment.

4.2 Who did what

The project was carried out collaboratively by two group members who worked together on all phases of the project. Both contributed equally to data preprocessing, feature extraction, model development, and evaluation. All decisions and implementations were shared responsibilities, and the most challenging aspects—such as optimizing model performance under limited computational resources and managing library compatibility issues—were faced and resolved jointly.

5. AI policies

During the completion of this project, AI assistance was used to refine the written content of the report. Specifically, OpenAI’s ChatGPT (GPT-4) was employed to improve the clarity, coherence, and academic tone of various sections.

The AI tool was not involved in any stage of model development, data processing, or experimental evaluation. Its role was limited to linguistic assistance, ensuring consistency in style and improving the quality of written explanations. All AI-generated suggestions were reviewed and manually verified to ensure factual accuracy and alignment with the project's implementation and outcomes.