# Predicting DW-Nominate Scores for Congressional Twitter Data

Max Mapstone and Jiaqi Yu

*Abstract*— **This study explores the use of congressional Twitter data to predict political ideology, represented by DW-NOMINATE scores, through various natural language processing techniques. We employed multiple text vectorization methods, including TF-IDF and Word2Vec, to convert tweets into structured data. Using AutoML, XGBoost, and other machine learning techniques, we developed predictive models to capture the relationship between tweet content and ideological leanings of congressional politicians. Our approach provides insights into how language patterns in social media can serve as predictors of political ideology, achieving promising results across different machine learning methods.**

## I. Introduction

DW-NOMINATE is a scaling method developed by political scientists Keith T. Poole and Howard Rosenthal to analyze voting patterns in legislatures, particularly the U.S. Congress. It uses multidimensional scaling to map legislators' roll-call votes onto a two-dimensional space, where the first dimension typically corresponds to the left-right (liberal-conservative) political spectrum. The second dimension sometimes accounts for other issues like race relations, though most congressional voting can be explained by the first dimension alone, due to the residual and often ambiguous nature of the second dimension. As a result, this second dimension is frequently downplayed in further research [Wikipedia, 2024]. DW-NOMINATE's ability to track ideological shifts over time makes it a crucial tool for studying political polarization and shifts in party dynamics.

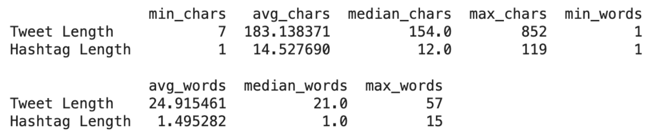
In our analysis of congressional Twitter data, we observed notable differences in tweet compositions between liberal and conservative ideologies. Conservative and liberal members often express opposing sentiments, particularly when addressing each other's policies, with a significant degree of negative sentiment directed at the opposing party's beliefs and practices. This aligns with broader trends in political polarization, where social media language increasingly reflects ideological divides.

Through the application of various natural language processing (NLP) techniques, including tf-idf and word2vec, we sought to predict DW-NOMINATE scores based on the linguistic patterns found in tweets. Key findings of our analysis showed that the average tweet length was 25 words, but through text cleaning and extraction of important terms, we reduced this to an average of 14 words. Additionally, our final model, using XGBoost in combination with tf-idf vectorization, achieved a root mean squared error (RMSE) of 0.26552. These results highlight the effectiveness of leveraging machine learning models to link social media data with traditional measures of political ideology, demonstrating the potential of modern data science techniques in political analysis.

## II. Data

Introduce your descriptive findings about the dataset here

In our analysis of congressional Twitter data, we aim to identify trends within the tweets that may aid in predicting DW-Nominate scores for the respective politicians. Notably, we discovered that each first dimension score corresponds uniquely to a second dimension score, and vice versa. To summarize the tweet composition, we created a table displaying the minimum, average, median, and maximum counts for the number of characters, words, and hashtags. Remarkably, the average tweet consisted of 25 words and 183 characters, while each tweet included approximately 1.5 hashtags, as illustrated in Figure 1.



```
                min_chars   avg_chars  median_chars  max_chars  min_words
Tweet Length            7  183.138371         154.0        852          1
Hashtag Length          1   14.527690          12.0        119          1

                avg_words  median_words  max_words
Tweet Length    24.915461          21.0         57
Hashtag Length   1.495282           1.0         15
```

Fig. 1. Tweet Summary Table

Examining the hashtags can yield further insights into predicting DW-Nominate scores, so we filtered the data to identify the ten most frequently occurring hashtags, shown in Figure 2. A significant number of these hashtags pertain to healthcare and the COVID-19 pandemic. Other prevalent hashtags relate to various government policies and opposing party viewpoints. By categorizing the data based on political affiliations, we observed notable differences among the groups. Groups 1 and 2 represented conservative tweets, while Groups 3 and 4 reflected liberal tweets. Common hashtags like COVID-19 appeared across both categories, but conservative groups tended to include hashtags such as "Obamacare" and "Tax Reform," whereas liberal groups featured hashtags like "Trumpcare" and "GOPTaxScam," as seen in Figures 3, 4, 5, and 6. Overall, it appears that tweets from all groups tend to express negative sentiments toward the opposing party's policies.

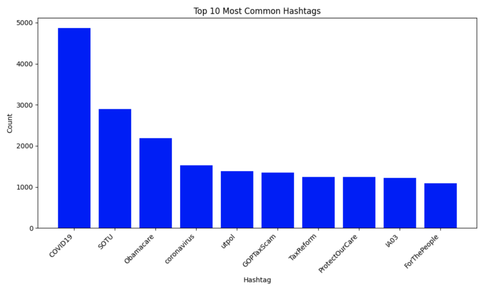


Fig. 2. Top 10 Hashtags

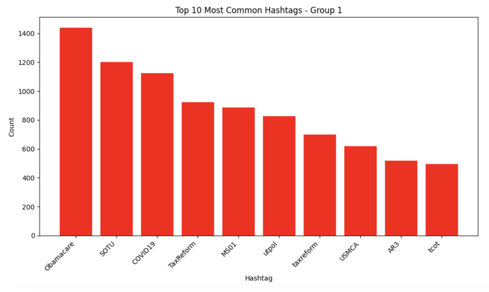


Fig. 3. Top 10 Hashtags for Group 1

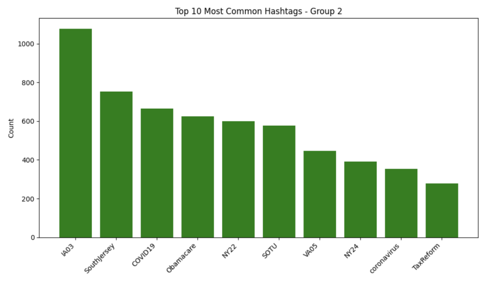


Fig. 4. Top 10 Hashtags for Group 2

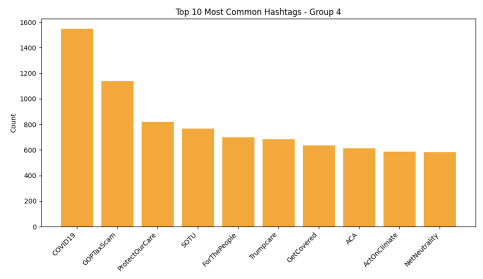


Fig. 5. Top 10 Hashtags for Group 3

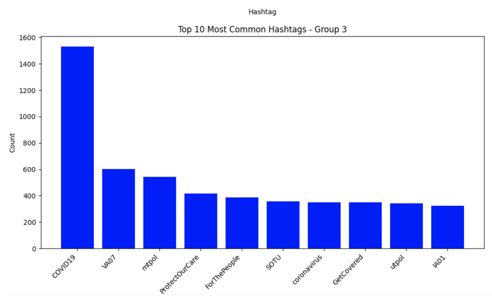


Fig. 6. Top 10 Hashtags for Group 4

To explore ideological shifts over time, we examined the tweets for their liberal or conservative leanings by year. By creating ridge-line plots, we could visualize these changes in the first dimension of the DW-Nominate scores over time. The analysis indicated slight ideological shifts, with conservative tweets trending "more conservative" and liberal tweets trending "less liberal," as depicted in the figure 8. Additionally, we assessed the distances between tweets that were most ideologically divergent to understand how their compositions varied. We identified the ten most distant tweets for both dimensions, as well as solely for the first and second dimensions. Many of these tweet pairs exhibited Euclidean distances of 2.3 (Dim 1) and 2.4 (Dim 2) from one another, with several extreme tweets appearing in multiple pairings, as shown in Figure 9.

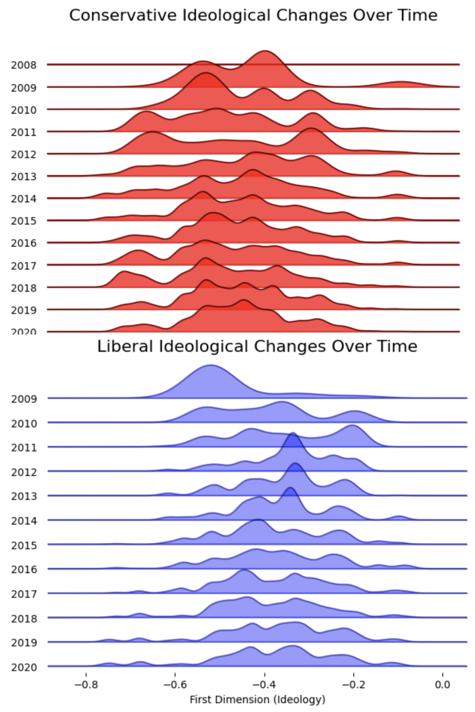
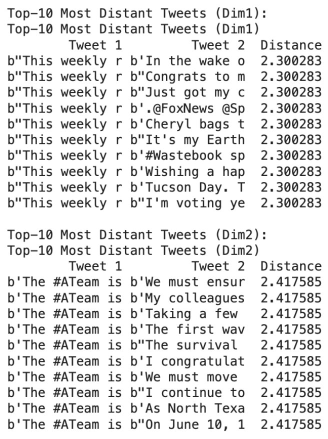


Fig. 8. Conservative and Liberal Ideological Changes over Time

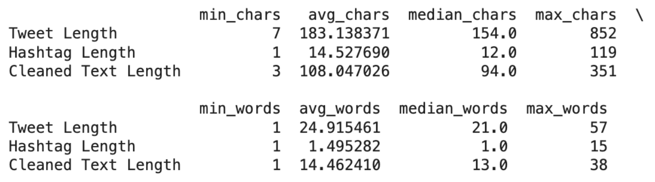


```
Top−10 Most Distant Tweets (Dim1):
Top−10 Most Distant Tweets (Dim1)
        Tweet 1          Tweet 2  Distance
b"This weekly r b'In the wake o  2.300283
b"This weekly r b"Congrats to m  2.300283
b"This weekly r b"Just got my c  2.300283
b"This weekly r b'.@FoxNews @Sp  2.300283
b"This weekly r b'Cheryl bags t  2.300283
b"This weekly r b"It's my Earth  2.300283
b"This weekly r b'#Wastebook sp  2.300283
b"This weekly r b'Wishing a hap  2.300283
b"This weekly r b'Tucson Day. T  2.300283
b"This weekly r b"I'm voting ye  2.300283

Top−10 Most Distant Tweets (Dim2):
Top−10 Most Distant Tweets (Dim2)
        Tweet 1          Tweet 2  Distance
b'The #ATeam is b'We must ensur  2.417585
b'The #ATeam is b'My colleagues  2.417585
b'The #ATeam is b'Taking a few   2.417585
b'The #ATeam is b'The first wav  2.417585
b'The #ATeam is b"The survival   2.417585
b'The #ATeam is b'I congratulat  2.417585
b'The #ATeam is b'We must move   2.417585
b'The #ATeam is b"I continue to  2.417585
b'The #ATeam is b'As North Texa  2.417585
b'The #ATeam is b"On June 10, 1  2.417585
```

Fig. 9. Top 10 Euclidean Distances between Tweets for each dimension

When integrating this data into a model, we needed to clean the text to eliminate unnecessary words and other elements that could complicate the analysis. Initially, we extracted mentions from the tweets where users tagged other accounts, as we believed this information, along with hashtags, constituted significant components of the tweets. We also removed any broken Unicode that represented potentially problematic characters. Additionally, links, emojis, and punctuation were eliminated due to their complexity for our models. We filtered out stop words and short words that contributed little meaning or sentiment to the overall tweet. Furthermore, we lemmatized the words to enhance the dataset. Post-cleaning, we observed improvements in the average character count, reducing it from 183 to 108, and the average word count, decreasing it from 25 to 14, as illustrated in Figure 10.



```
                     min_chars   avg_chars  median_chars  max_chars  \
Tweet Length                 7  183.138371         154.0        852
Hashtag Length               1   14.527690          12.0        119
Cleaned Text Length          3  108.047026          94.0        351

                     min_words  avg_words  median_words  max_words
Tweet Length                 1  24.915461          21.0         57
Hashtag Length               1   1.495282           1.0         15
Cleaned Text Length          1  14.462410          13.0         38
```

Fig. 10. Updated Tweet Summary Table

Utilizing unsupervised learning techniques was also instrumental in analyzing this Twitter data, as we sought to identify ten topics present within the tweets. Using Latent Dirichlet Allocation (LDA), we extracted ten topics from the tweet contents; however, the method was slow, particularly with a maximum of five iterations, leading us to reduce it to two iterations. This resulted in less convergence and suboptimal topic distinction. Subsequently, we employed Nonnegative Matrix Factorization (NMF), which generated more distinct topics, such as healthcare, economy, and education. This model provided clearer and more defined topics, both methods are demonstrated in Figures 11 and 12 respectively.

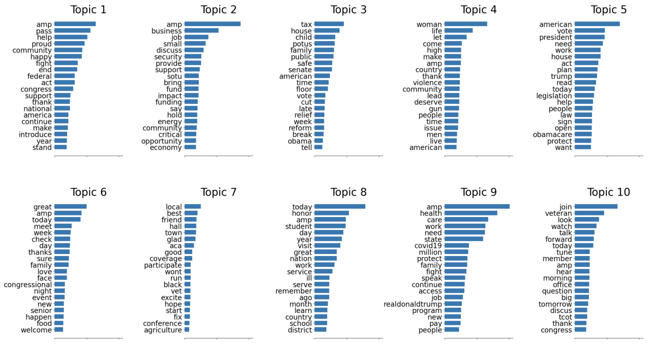


Fig. 11. LDA Topics

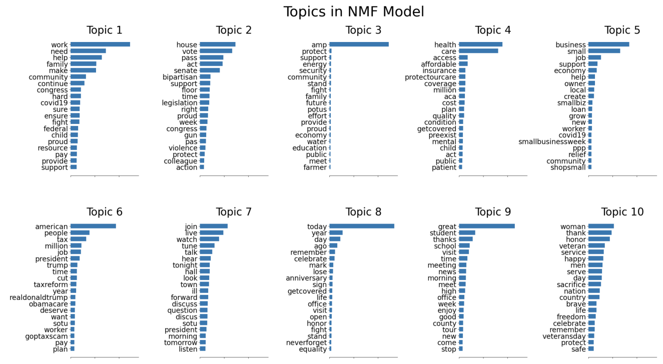


Fig. 12. NMF Topics

## III. Methods

### A. NLP

In this project, we applied three Natural Language Processing (NLP) techniques to extract text-based features: Word2Vec, TF-IDF, and BERT.

*1) Word2Vec:* At the beginning we thought of Word2Vec[Jatnika et al., 2019], which is the easiest way of extracting features from text. Word2Vec was used to generate vectorized representations of words from the textual data. It uses a shallow neural network to map words to vectors in a continuous vector space. Each word gets mapped to a dense vector, and the similarity between words can be inferred from their distances in this space.

However, In our approach, Word2Vec was found to be time-consuming and prone to overfitting when used without careful hyperparameter tuning, such as the number of features (vector dimensions) and the size of the context window. And due to insufficient submissions and bad results(nearly 0.35 RMSE), we had to give up Word2Vec.

*2) TF-IDF:* After Word2Vec we tried TF-IDF[Qaiser and Ali, 2018] and BERT[Koroteev, 2021]. The Term Frequency-Inverse Document Frequency (TF-IDF) method was taught in Data Mining class. It converts text to numerical features by calculating how important a word is in a document relative to the entire dataset. The text in each document was converted into a x-dimensional feature vector using the most frequent words.

Compared with Word2Vec and BERT, TF-IDF is relatively much faster, which saves us a lot of resources.

*3) BERT:* Bidirectional Encoder Representations from Transformers (BERT) was used to generate contextual embedding for the text. Unlike Word2Vec and TF-IDF, BERT captures the meaning of words based on their context in the sentence. However, BERT requires significant computational resources and time, especially on large datasets.

In this project, we found that BERT, when paired with XGBoost and other models, provided competitive results but required careful handling of GPU resources to avoid memory bottlenecks(which caused us fail to generate a model).

### B. Machine Learning

We have tried various machine learning algorithms, each with its own strengths and weaknesses, as outlined below.

Number of features, window size, and training time/epochs were key factors influencing the performance of the models.

*1) H2O AutoML:* AutoML was used to automate the process of model selection and hyperparameter tuning[LeDell and Poirier, 2020]. Although it's a convenient method, AutoML cannot fully utilize GPU resources, which makes it slower compared to more direct algorithms like XGBoost. It is also time-bound rather than epoch-bound, meaning training is limited by a time duration rather than a specific number of iterations. While it finds competitive models, it can take a long time to find optimal solutions, especially for large datasets.

*2) XGBoost:* XGBoost was one of the most effective models in this project[Chen and Guestrin, 2016]. It was run on GPUs, taking advantage of parallel processing to reduce training time. The model performance was directly tied to the number of epochs and the early stopping criterion, which prevented overfitting. XGBoost showed better generalization when tuned with parameters such as max depth, min child weight, and learning rate. Given that it utilizes GPU resources well, XGBoost provided good trade-offs between performance and speed in this project.

*3) Others:* Other machine learning methods such as TPOT and ensemble algorithms were also tested[Olson and Moore, 2016]. However, TPOT encountered issues, likely due to insufficient preprocessing or the large memory requirements of the model-building process. These issues caused frequent crashes, which made TPOT impractical in our experiments.

## C. Steps

*1) Preprocessing:* For all the models, preprocessing involved tokenizing the text data, applying TF-IDF, and other feature extraction techniques like Word2Vec or BERT embeddings. Additionally, non-text features like favorite count, retweet count, and year were normalized and incorporated into the models. Special attention was given to handling missing values by filling them with empty strings and ensuring no invalid entries in the data.

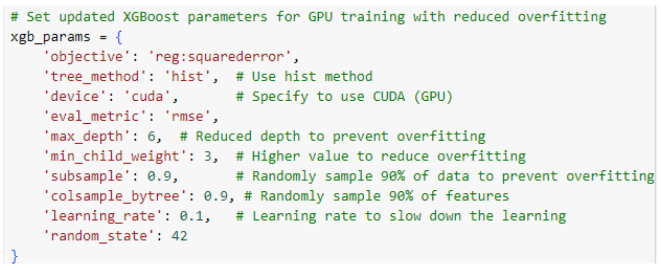
*2) Model Selection:* Based on the task's nature and the available computational resources, we selected several models, including AutoML, XGBoost, and TPOT.

| | **H2O AutoML** | **XGBoost** |
|---|---|---|
| **Training time** | 4h each dimension, CPU | 5min with A100 GPU |
| **RMSE** | minimum: 0.32 | minimum: 0.27 |

TABLE I
COMPARISON OF H2O AUTOML AND XGBOOST

Each model was tested for its efficiency in both time and resource consumption. As shown in Table 1, XGBoost was selected as the best-performing algorithm in terms of speed and resource utilization, especially when using GPU-accelerated training.

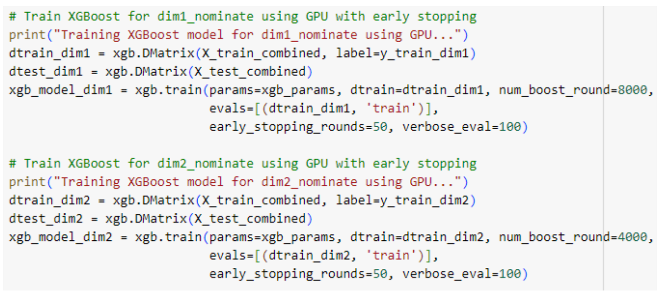
*3) Model Evaluating and Parameter Adjustment:* The evaluation of models was done using RMSE as the primary metric. For each model, parameters such as learning rate, max depth, subsample ratio, and early stopping were tuned then fixed based on several cross-validation results.



```
# Set updated XGBoost parameters for GPU training with reduced overfitting
xgb_params = {
    'objective': 'reg:squarederror',
    'tree_method': 'hist',  # Use hist method
    'device': 'cuda',       # Specify to use CUDA (GPU)
    'eval_metric': 'rmse',
    'max_depth': 6,  # Reduced depth to prevent overfitting
    'min_child_weight': 3,  # Higher value to reduce overfitting
    'subsample': 0.9,       # Randomly sample 90% of data to prevent overfitting
    'colsample_bytree': 0.9, # Randomly sample 90% of features
    'learning_rate': 0.1,   # Learning rate to slow down the learning
    'random_state': 42
}
```

Fig. 13. Fixed Machine Learning Parameters

Figure 8 shows the fixed parameters while building multiple models, they decide the speed and extent of training. Figure 9 shows the Adjustable parameters, which includes feature size, number of epochs, and early stopping rounds[Brownlee, 2024][Paperspace Blog, 2024]. In the case of XGBoost, early stopping was used to avoid overfitting by halting training when the performance on the validation set no longer improved. Also the number of epochs for two dimensions are set differently, when we found that dimension 1 is hard to converge.



```
# Train XGBoost for dim1_nominate using GPU with early stopping
print("Training XGBoost model for dim1_nominate using GPU...")
dtrain_dim1 = xgb.DMatrix(X_train_combined, label=y_train_dim1)
dtest_dim1 = xgb.DMatrix(X_test_combined)
xgb_model_dim1 = xgb.train(params=xgb_params, dtrain=dtrain_dim1, num_boost_round=8000,
                           evals=[(dtrain_dim1, 'train')],
                           early_stopping_rounds=50, verbose_eval=100)

# Train XGBoost for dim2_nominate using GPU with early stopping
print("Training XGBoost model for dim2_nominate using GPU...")
dtrain_dim2 = xgb.DMatrix(X_train_combined, label=y_train_dim2)
dtest_dim2 = xgb.DMatrix(X_test_combined)
xgb_model_dim2 = xgb.train(params=xgb_params, dtrain=dtrain_dim2, num_boost_round=4000,
                           evals=[(dtrain_dim2, 'train')],
                           early_stopping_rounds=50, verbose_eval=100)
```

Fig. 14. Adjustable Machine Learning Parameters

## IV. RESULTS

After conducting the experiments, the **XGBoost** model provided the best performance with an RMSE value of **0.26552** on the test set. This was significantly better than the H2O AutoML model, which took longer to train and didn't utilize GPU resources at all.

However, the results could have been further improved with more computational resources, especially when using BERT combined with XGBoost or TPOT. The TPOT model could not be successfully built due to resource constraints and memory limitations, which points to potential issues in our preprocessing pipeline or model configuration.

Overall, the project demonstrated that XGBoost is highly efficient when GPU resources are available, offering both faster training and superior performance compared to other models. However, handling more complex text embeddings like BERT still presents challenges in resource-constrained environments, and we are very interested in finding out how BERT and TPOT does in this assignment.

## References

[Brownlee, 2024] Brownlee, J. (2024). Xgboost with python mini-course. https://machinelearningmastery.com/xgboost-with-python-mini-course/. Accessed: 2024-06-22.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

[Jatnika et al., 2019] Jatnika, D., Bijaksana, M. A., and Suryani, A. A. (2019). Word2vec model analysis for semantic similarities in english words. *Procedia Computer Science*, 157:160–167.

[Koroteev, 2021] Koroteev, M. V. (2021). Bert: A review of applications in natural language processing and understanding. *arXiv preprint arXiv:2103.11943*.

[LeDell and Poirier, 2020] LeDell, E. and Poirier, S. (2020). H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020. ICML San Diego, CA, USA.

[Olson and Moore, 2016] Olson, R. S. and Moore, J. H. (2016). Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on Automatic Machine Learning*, pages 66–74. PMLR.

[Paperspace Blog, 2024] Paperspace Blog (2024). A comprehensive guide to xgboost. https://blog.paperspace.com/xgboost-optimization-tips/. Accessed: 2024-07-18.

[Qaiser and Ali, 2018] Qaiser, S. and Ali, R. (2018). Text mining: Use of tf-idf to examine the relevance of words to documents. *International Journal of Computer Applications*, 181(1):25–29.

[Wikipedia, 2024] Wikipedia (2024). Nominate (scaling method). https://en.wikipedia.org/wiki/NOMINATE_(scaling_method). Accessed: 2024-09-05.