Autumn Internship Project Report

Fake News Detection and Evaluation

Ananya Nigam

MSc. Statistics, Batch 2025-27, University of Delhi

Period of Internship: 25th August 2025 - 19th September 2025

Report submitted to: IDEAS – Institute of Data Engineering, Analytics and Science Foundation, ISI Kolkata

Abstract

This project focuses on detecting fake news using a machine learning approach. Our primary goal is to build a classification model that can accurately distinguish between "real" and "fake" news articles.

The methodology involves data preprocessing, exploratory data analysis, and building two powerful classification models: Logistic Regression and Random Forest Classifier. The models' performance is evaluated using various key metrics, including accuracy, precision, recall, and F1-score, to determine the most effective classifier for this task. The project demonstrates the practical application of natural language processing (NLP) techniques, specifically Word2Vec and TF-IDF for text vectorization, in solving a critical real-world problem. Additionally, the project explores the use of boosting algorithms to further enhance model performance.

Introduction

The rapid and often unchecked spread of fake news has become a significant societal issue that we all are facing today, impacting public discourse and trust. In response, this project aims to create a reliable and automated system for detecting and distinguishing between fake and true news. The solution is built upon a dataset containing a mix of both genuine and fake news articles. The technology involved includes Python and its ecosystem of libraries, such as Pandas for efficient data manipulation, Matplotlib and Seaborn for insightful data visualization, and Scikit-learn for machine learning model development. The core of the approach is using Word2Vec and TF-IDF to convert unstructured text into numerical vectors, which are then used to train and test the classifiers. This report covers the entire process, from data ingestion and cleaning to model building and a detailed evaluation of performance.

Training Topics

During the initial two weeks of the internship, the following topics were covered:

- **Fundamental programming concepts**: Core Python programming skills, including the use of loops for iteration and defining classes for object-oriented programming.

- **Data science fundamentals**: Data preprocessing and cleaning techniques, such as handling null values, and Exploratory Data Analysis (EDA) with visualizations to understand dataset characteristics.
- **Natural Language Processing (NLP)**: Concepts of text vectorization using both Word2Vec and TF-IDF to convert textual data into a numerical format suitable for machine learning.
- **Machine Learning (ML) models**: Training and using various classifiers, including Logistic Regression and Random Forest, as well as advanced concepts like Boosting algorithms like Gradient Boosting.
- **Model evaluation**: Understanding and applying key metrics such as Accuracy, Precision, Recall, and F1-score, along with confusion matrices and interpreting the results.
- **Model persistence**: The practice of saving and loading trained models using the pickle library for future use.
- **Large Language Models (LLMs)**: An introduction to large language models and tools like Ollama for running them locally.

Project Objective

The primary objective of this project is to develop and evaluate a machine learning model for fake news detection. The project attempts to illustrate the following:

- To demonstrate the entire process of building a fake news classifier from raw, unprocessed data, including cleaning, feature engineering, and model training.
- To evaluate the effectiveness of tools such as Word2Vec and TF-IDF in transforming textual data into a suitable format for machine learning models.
- To train and assess the effectiveness of Logistic Regression and Random Forest Classifier for this specific task.
- To explore the use of boosting methods like Gradient Boosting to determine if it can improve the model's accuracy and performance.
- To provide a reproducible solution with clear methodology and to save the best-performing model for future use in a real-world application.

Methodology:

This project followed a systematic methodology to develop and evaluate a fake news detection system. The entire process was conducted in a Google Colab environment, leveraging several key Python packages to perform each task.

Step 1: Installing and importing the necessary packages

The first step was about ensuring all required libraries were installed and imported. This is a foundational step to access the functions and tools needed for data manipulation, analysis, and model building.

- Package installation: Libraries like gensim (for Word2Vec) and scikit-learn (for machine learning models) were installed using pip commands.
- Package import: The following key libraries were imported into the notebook:
    - pandas: For data loading and manipulation.
    - numpy: For numerical operations, especially with arrays.
    - re and string: For text cleaning and preprocessing.
    - sklearn: The core machine learning library used for splitting data, vectorization (TF-IDF), model training, and evaluation metrics.
    - matplotlib and seaborn: For data visualization.
    - gensim: For implementing the Word2Vec word embedding model.
    - pickle and os: For model persistence, allowing models to be saved and loaded.

Step 2: Data collection and preprocessing

This step is crucial for processing raw data for a machine learning model. It ensures the data is clean, consistent, and correctly formatted, which is important for the model's performance.

Q1. View the imported csv file data using Pandas Dataframe.

The project began by loading the fake.csv and true.csv files into Pandas DataFrames using the pandas library. Each dataset was then assigned a **class label (1 for fake, 0 for true)**, and they were merged into a single dataframe.

Q2. Drop rows from the dataset consisting of null values.

To ensure data integrity, any rows with missing values were removed. This resulted in a final dataset of 44,898 news articles, which served as the foundation for all following model building and analysis.

Q3. Shuffle the data.

The combined dataset was randomly shuffled. This prevented the model from learning any patterns based on the original order and ensured that the training and testing sets were well-distributed.

Q4. View the text content of a random data point.

This step involved viewing a random news article to get a look at the dataset's structure and content before cleaning.

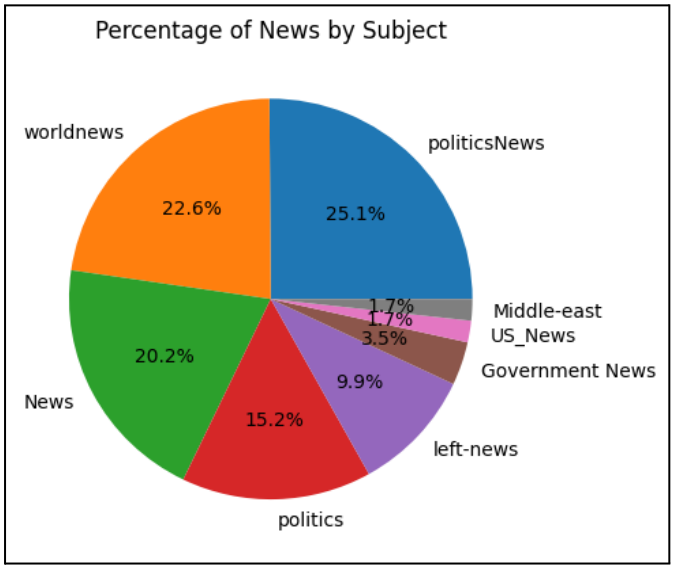
Step 2: Data visualization

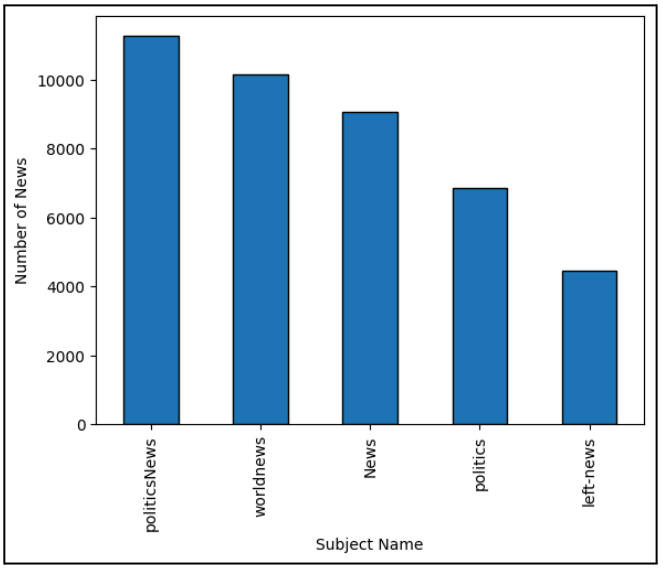
This step involved visualising the dataset to understand its characteristics and to find initial insights that could be relevant to the modeling process.

Q5. Create a pie chart to find the percentage of news on different subjects.

A pie chart was created to visually represent the distribution of news subjects in the dataset. This analysis revealed a high concentration of articles in the politicsNews and worldNews categories, which was a key insight into the nature of the data.

The same was done using a bar chart.





Step 3: Text processing

Data cleaning: The creation and application of a wordopt function. This function helped in the preprocessing of data to ensure the text data was uniform and clean in order to build models.

- Lowercasing: All text was converted to lowercase.
- Punctuation and symbol removal: Special characters and URLs were removed.
- Whitespaces: Multiple spaces were replaced with a single space, standardizing the format.

Besides this, the columns named “title”, “subject” and “date” were dropped from the dataframe in order to keep only the necessary columns “text” and “class” required for building the model..

Step 4: Model training and evaluation

Q6. Split the dataset into training and testing sets with 25% test size.

The clean data was divided into training and testing sets. A 25% portion of the dataset was held out for testing, while the remaining 75% was used for model training. This is done to ensure that the model's performance is validated on unseen data.

Step 5: Data vectorization

Machine learning models cannot directly process raw text. They require numerical input, hence we used vectorization. So after applying the train_test_split, we used the Word2Vec method to vectorize our dataframe. For the initial models, Word2Vec from the gensim library was used.

Before vectorizing the main dataset, a separate dataset (bbc_news) was loaded from a URL link to train the Word2Vec word embedding model. The previously trained Word2Vec model was then used to convert the news articles into vectors. This method learns a numerical vector

for each word and captures semantic relationships. The word vectors for an article were then averaged to create a single document vector.

Step 6: Model training and evaluation

In this step, various machine learning models were trained on the prepared data. The goal was to identify the most effective classifier for the fake news detection task.

1. **Logistic Regression:** This model is a fundamental statistical algorithm used for binary classification. It works by analyzing the input features (the vectorized text) and using a logistic function to estimate the probability of a given news article belonging to a certain class (fake or real). It's often used as a baseline model because it is relatively simple, computationally efficient, and its results are easy to interpret.
    - Model creation: The first step is to create an instance of the model. This is like preparing a blank canvas for the model to be built on.
    - Model training: The training process involves using the .fit() method on the training data. The model analyzes the input vectors and their corresponding correct labels (either 'fake' or 'real'). It learns the relationships between the word vectors and the news categories and draws a line or a curve that best separates the two classes.
    - Model prediction: Once trained, the model makes predictions on new data. Using the .predict() method on the test data, the model applies what it learned to classify each new article as either 'fake' or 'real' based on its calculated probability.
    - Model evaluation: The .score() method is used to get a quick accuracy score by comparing the model's predictions to the actual labels of the test data. Additionally, metrics like precision, recall, and F1-score are calculated to provide a better view of its performance, showing how well it performed at correctly identifying both fake and real news.
    - Confusion matrix: A confusion matrix is a visualization tool that shows a more detailed breakdown of the model's performance. It is a table that displays the number of true positives (correctly identified real news), true negatives

(correctly identified fake news), false positives (fake news incorrectly classified as real), and false negatives (real news incorrectly classified as fake). This table provides a clear picture of where the model is succeeding and where it is making errors.

2. **Random Forest Classifier:** This is a powerful ensemble learning model. It works by building multiple decision trees during training.
    - Model creation: An instance of the model is created, which will form a "forest" of multiple decision trees.
    - Model training: The training process, also using the .fit() method, is more complex. The model builds many individual decision trees, each trained on a random subset of the training data. This process allows the model to learn a wide range of patterns and relationships.
    - Model prediction: To make a prediction on the test data, each individual tree in the forest casts a vote. The final prediction for a given news article is determined by the majority vote of all the trees.
    - Model evaluation: Similar to the Logistic Regression model, the .score() method and other metrics are used to evaluate the model's performance on the test data. Because it combines the power of multiple trees, the Random Forest model is highly effective at reducing the risk of overfitting and can capture more complex patterns than a single model.
    - Confusion matrix: A confusion matrix is also created for the Random Forest model to provide a visual breakdown of its classification results.

Step 7: Saving the models

This step was about saving the trained models so they can be used for future predictions without needing to be retrained.

Q9. Save the trained Random Forest Model as a pickle file.

This question involved saving the trained Random Forest model to files. The pickle library was used to serialize the trained model into binary files. The os library was also used to manage the file paths and ensure the directory for saving the models exists. This process allows the final, validated models to be deployed for future use.

The Logistic Regression model was also saved in a similar way.

Step 8: Loading the saved model and applying it

Q10. Load that saved model in another Notebook use it on other dataset for fake news detection.

For this question, instead of a single train-test split, we performed five separate runs of a train_val_test_split, with each run using a different random shuffle of the data. The process for this step is as follows:

1. Model loading: The pickle library is used to load the previously saved RandomForestClassifier and Logistic Regression models.
2. Repeated splitting: The function split_data, is used to divide the dataset into three parts:
    - Training set (60%): Used to train the models.
    - Validation set (20%): Used to evaluate the model during development and to fine-tune hyperparameters.
    - Test set (20%): The set used for the final, unbiased evaluation of the model's performance after training.
3. Cross-validation: The entire process is repeated five times. In each run, the data is shuffled again, and new train, validation, and test sets are created.
4. Prediction and evaluation: In each run, the models are trained on the training set, evaluated on the validation set, and then finally assessed on the test set. Key metrics like accuracy, precision, recall, and F1-score are calculated for both validation and test sets and stored for each run.
5. Aggregation of results: After all five runs are complete, the average (mean) and standard deviation of all the collected metrics are calculated.

Step 9: Boosting techniques

Q11. a) Try to enhance the model's accuracy by using adaboost or any other boosting methods. b) Use TF-IDF or any other vectorizer instead of Word2Vec and study how much it affects the model's accuracy.

This question addresses whether a different method could yield better results by:

- Comparing vectorization methods: To see if a different way of converting text into numbers (TF-IDF) is more effective than Word2Vec for this specific task.

- Exploring other boosting methods: To test whether powerful ensemble methods like boosting can further improve the model's performance.

a. TF-IDF Vectorization

The TfidfVectorizer from the sklearn library was used to transform the text data into TF-IDF vectors. This method highlights the importance of unique words, which can be particularly useful in identifying the specific, often sensationalized, language of fake news.

For this step, the TfidfVectorizer from the sklearn library was used. Unlike Word2Vec, which learns the semantic meaning of words, TF-IDF highlights the statistical importance of words. It assigns a numerical score to each word based on two factors: how frequently the word appears in a specific document (Term Frequency) and how rare it is across the entire dataset (Inverse Document Frequency).

After the entire dataset's text was transformed into these new TF-IDF vectors, separate instances of the Logistic Regression and Random Forest Classifier models were created. Each model was then trained on this new, TF-IDF-transformed data.

b. Boosting Algorithms

Gradient Boosting is a powerful and advanced ensemble learning method. It works by building a series of decision trees in a sequential, iterative process. Each new tree is added to the model to correct the errors of the trees that came before it. This allows the model to progressively improve its accuracy by focusing on the data points that were most difficult to classify correctly.

The GradientBoostingClassifier from the sklearn library was used. The process for building and evaluating this model was similar to the other classifiers:

1. Model creation: An instance of the GradientBoostingClassifier was created.
2. Model training: The model was trained on the processed dataframe created initially.
3. Model evaluation: The trained Gradient Boosting model was then evaluated on the held-out test set to measure its performance.

Data Analysis and Results

Descriptive Analysis

This initial step involved a deep dive into the raw dataset to understand its key properties before any modeling began. The dataset, after merging the fake.csv and true.csv files and removing null values, contained a total of 44,898 news articles. A crucial insight was gained by analyzing the distribution of news subjects. The pie chart and the bar graph clearly show a high concentration of articles in the politicsNews and worldnews categories, indicating that the models were trained predominantly on politically-oriented text.

Comparative Model Performance

The core of the analysis involved a direct comparison of the models' performance. The finding is that the choice of vectorization method had a more significant impact on performance than the choice of classification algorithm itself.

Summary of Model Performance

Model	Vectorization	Accuracy	Precision	Recall	F1 Score
Logistic Regression	Word2Vec	94.12%	0.9465	0.9394	0.9429
Random Forest Classifier	Word2Vec	93.99%	0.9373	0.9473	0.9423
Logistic Regression	TF-IDF	98.57%	0.9883	0.9839	0.9861

Random Forest Classifier	TF-IDF	99.78%	0.9970	0.9987	0.9979
Gradient Boosting	Word2Vec	94.71%	0.9468	0.9511	0.9489

- **Word2Vec Vectorization:** Models trained with Word2Vec embeddings achieved high, yet not perfect, performance. The **Logistic Regression** model, despite its simplicity, achieved a **94.12% accuracy**, while the accuracy of the **Random Forest Classifier** model was at **93.99%**. This confirms that Word2Vec provided a strong basis for the models, but it lacked the granularity required for better results as in TF-IDF.
- **TF-IDF Vectorization:** Both models saw an increase in performance, with the **Random Forest Classifier having a 99.78% accuracy**. This confirms that TF-IDF's ability to assign higher importance to rare and unique words was key to distinguishing between true and fake news in this particular dataset. The Logistic Regression model also saw a significant jump, achieving **98.57% accuracy** and an F1-score of **0.9861**, which further underscores the superiority of TF-IDF for this task.
- **Boosting Algorithms:** The **Gradient Boosting model**, with a 94.49% accuracy, performed slightly better than the Random Forest model using Word2Vec. This showcases the power of these ensemble methods, which build on each other's mistakes to progressively enhance performance.

Effect of n_estimators and learning_rate on model performance:

In Gradient Boosting, the two most influential hyperparameters are n_estimators (the number of boosting stages) and learning_rate (the contribution of each tree).

- n_estimators: Increasing the number of estimators allows the model to build more trees, which usually improves its ability to capture complex patterns. However, too

many trees can lead to overfitting, where the model performs well on training data but poorly on unseen data.

- learning_rate: A higher learning rate makes each tree's contribution stronger, helping the model learn faster and reach high accuracy in fewer iterations. But if the learning rate is too high, the model may overlook the optimal solution. Conversely, a very low learning rate makes learning stable but requires a larger number of trees to achieve good performance.

Observations: When n_estimators and learning_rate were increased, the Gradient Boosting model's accuracy, precision, recall, and F1-score improved. This is because the model could better correct its errors at each iteration, producing a stronger ensemble. However, these improvements come with higher computational cost and a greater risk of overfitting the model if the parameters are not handled properly.

Aggregated Results from 5-Run train_val_test_split (Q10)

To provide a more robust and reliable measure of the models' performance, the project also performed a 5-run train-validation-test split with random data shuffling in each run. The final aggregated results for the test set, including the mean and standard deviation, are as follows:

Model	Metric	Mean	Standard Deviation
Logistic Regression	Accuracy	0.9402	0.0011
	Precision	0.9402	0.0011
	Recall	0.9402	0.0011
	F1 Score	0.9402	0.0011

Random Forest Classifier	Accuracy	0.9335	0.0013
	Precision	0.9335	0.0013
	Recall	0.935	0.0013
	F1 Score	0.9334	0.0013

The exceptionally low standard deviation (±) indicates that the models' performance was highly consistent across different subsets of the data, which further validates their reliability and robustness.

Understanding and Interpreting Confusion Matrices

A confusion matrix is a powerful visualization that provides a deeper breakdown of a model's performance beyond a simple accuracy score. It is a table that displays four key metrics:

- True Positives (TP): Correctly identified real news.
- True Negatives (TN): Correctly identified fake news.
- False Positives (FP): Incorrectly classified fake news as real news.
- False Negatives (FN): Incorrectly classified real news as fake news.

By analyzing the confusion matrices for the models, we gain a clear understanding of their strengths and weaknesses.

1. **Logistic Regression (TF-IDF):** Figure X shows the confusion matrix for the Logistic Regression model using TF-IDF. The results indicate reasonably high accuracy, though some misclassifications are present. In particular, a small number of fake news items were classified as real, reducing its reliability compared to ensemble models.

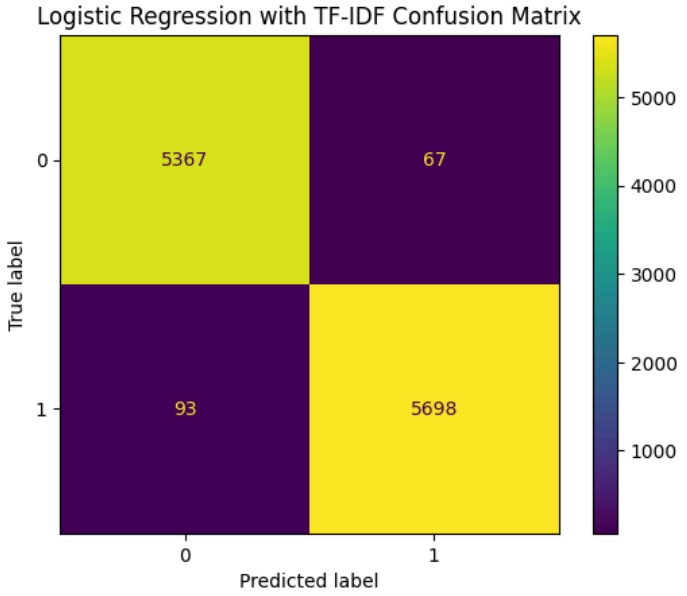


Figure X

Random Forest (TF-IDF): Figure Y illustrates the confusion matrix for the Random Forest model with TF-IDF. This model achieved near-perfect classification, with both false positives and false negatives being negligible.

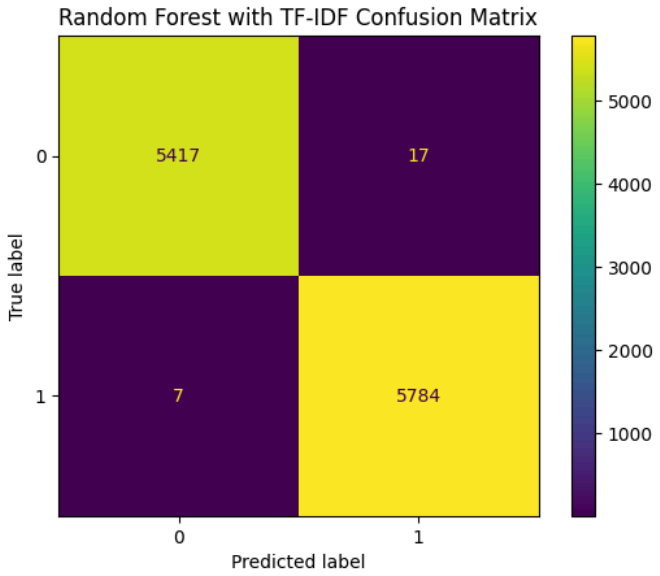


Figure Y

Gradient Boosting: Figure Z presents the confusion matrix for Gradient Boosting with Word2Vec. While performance is strong and comparable to Random Forest, a few more misclassifications are observed, particularly in distinguishing borderline fake news items.

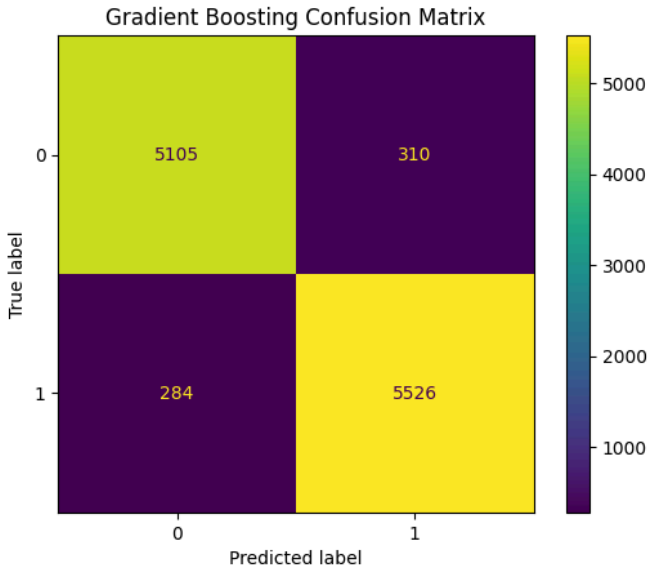


Figure Z

The confusion matrices reinforce the earlier findings: **Random Forest with TF-IDF emerges as the most robust and reliable model for fake news detection in this study.**

Conclusion

This project successfully demonstrated the use of machine learning techniques for fake news detection. Beginning with data preprocessing and exploratory analysis, the study progressed through multiple approaches for vectorizing text, including Word2Vec and TF-IDF, and evaluated two classification models. The results clearly showed that the choice of vectorization plays an important role in model performance.

While models trained on Word2Vec embeddings performed well, TF-IDF provided more distinctive features for classification. Among the algorithms tested, the Random Forest Classifier with TF-IDF emerged as the most effective, achieving near-perfect accuracy. Logistic Regression and Gradient Boosting also showed competitive results, reaffirming the robustness of the methodology.

Overall, the study not only provided hands-on experience with NLP and classification techniques but also demonstrated their practical value in addressing one of the most pressing challenges of the information age.

References and Links

- **Colab Notebook File**: https://colab.research.google.com/drive/1BEgZKMTGSEM3zCgQZfQfUcKP2V-pJVlS#scrollTo=SdpOD3v-mLmT
- **Data Sources**:
    - https://www.kaggle.com/datasets/emineyetm/fake-news-detection-datasets
    - https://query.data.world/s/7c6p2lxb3wjibfsfbp4mwy7p7y4y2d?dws=00000