VERACITY

A MULTI-DIMENSIONAL NEWS ANALYZER AND SCORER

With a Focus on fake-news Detection

Submitted By : **Agrima Jain**

Matriculation Number: [redacted]

Submitted for the Course**: Natural Language Processing WS25/26**

Submitted To: **Professor Dr.** [redacted]

Submitted On: **23.01.2026**

Table of Contents

ABSTRACT

Fake news has become one of the most common issues online as compared to before. Fake news, mostly spreading through word-of-mouth before social media platforms, has now especially increased with technological advancements in online communication. Surprisingly, social media is not the only contributor of and medium for spreading false information. More so, in fact, are also dozens of online tabloids, websites, and blogs responsible for the origin and propagation of false information. The fast flow of information and large number of users connected online is what supports publication of false news giving it the potential to reach many people. Manual fact-checking is immensely impractical at this scale and so there is a need for development of more sophisticated, automated systems.

This paper details the design, development, and implementation of VeraCity, a multi-dimensional news analyzer and scoring framework. The paper is centered on the explainable artificial intelligence (XAI) prototype of VeraCity, which was created to identify misinformation using a hybrid approach of statistical machine learning (ML) and rule-based linguistic heuristics. The system combines linear models and vectorization by incorporating the techniques of Support Vector Machines (SVM) and Term-Frequency Inverse Document Frequency (TF-IDF) along with a linguistic analysis of subjectivity and emotion-intensity in texts for developing a machine learning plus rule-based system.

INTRODUCTION

The inability to clearly define a boundary between truth and fiction can damage trust in public institutions, sciences (through spread of medical misinformation and hoaxes) and journalism. The impact is not limited to only digital forums; there are physical consequences. For example, during the COVID-19 pandemic, there were hundreds of (preventable) deaths from drinking alcohol-based cleaning products which was spread as false information and wrongly believed to be a cure for the virus. (Coleman, 2020).

Similarly, the July 2024 Southport stabbing demonstrates how unverified claims about the suspect circulated widely on social media due to the rapid sharing of a now-deleted LinkedIn post. The false claim reached millions of users within hours and was repeatedly presented as “confirmed” evidence. By the time accurate details were released, protests had already escalated into riots; a perfect example of how quickly false information can spread and produce real-world consequences before corrections take effect. (Sardarizadeh, 2024) False news travels faster than the truth because it is sometimes designed to trigger strong emotional responses (like outrage, fear, distress). An automated news-reliability scorer or checker could have helped flag such claims as low-credibility and helped prevent people from blindly trusting such rumors.

It is not always easy to identify news in a binary way, that is, to classify information as completely fake or completely true. The challenge in binary classification arises due to the increasing degree of stylistic overlap between credible reports and misinformation, because modern fake news often adopts the writing style of real-news reporting therefore evading detection. Unfortunately, fake news, is often mixed with professionally written, long, and referenced data that allows the reader to more easily agree with and believe the incorrect information presented. Indicators (words & phrases) of real news have started to make their way into fake statements and reports.

Binary Classification is becoming increasingly insufficient. A text can be factually true, but biased or it can be unbiased but rely on unverified (unreliable) data with misleading, false content. To accurately assess information in a complex digital space, a system should evaluate dimensions of a text as individual components too, like stylistic intent, inherent bias, emotional intensity, and linguistic patterns. Utilizing such structural indicators, can better identify the subtle distinctions between objective reporting, emotionally-charged reporting and biased information.

RELATED WORK

Survey of Existing Work

Pérez-Rosas et al. (2018), in the paper, "Automatic Detection of Fake News" explore the linguistic patterns that distinguish fake news from real news by analyzing various stylistic features across multiple datasets. The research relies on differences in punctuation, psycholinguistic features (links between language and cognition, emotion, and social behavior) using the LIWC Lexicon, readability (through number of characters, long words, complex words, number of syllables, word types, and number of paragraphs), and syntactic features as distinguishers of deceptive content. Their paper focuses on the first major category of disinformation—comprising **serious fabrications, hoaxes, and satire**—across two domains: celebrity gossip and six other specific news groups." The paper utilizes a liner SVM, similar to the core model used in this prototype. Their research also focused on cross-domain application and analysis of their detection model. By leveraging datasets with news spanning multiple topics, they carried out a cross-domain detection of fake news. The results showed a significant loss in the accuracy of cross-domain applications as compared to within-domain. This paper contains strong theoretical foundations for the concepts of subjectivity and polarity scores and linguistic risk used in this prototype, VeraCity.

Shu et al. (2020) provides the construction of a multi-dimension data repository *FakeNewsNet,* containing two datasets with news content, social context, and dynamic information. The paper states that in addition to analyzing news content, extra information like social engagements and social behaviors of users should also be explored. It explains how since fake news is likely to be created and spread by non-human social media accounts such as bots, it is essential to capture user profiles. They analyze the news content itself based on the distribution of news and distribution of publishers publishing fake news. While the social context of their data consists of information regarding user profiles, user posts, and network structures (social networks). The dynamic information in their dataset reflects temporal user engagement for news articles. Their paper highlights fake news detection, fake news evolution, and fake news mitigation as potential applications of the *FakeNewsNet* dataset created. Although, not considering social aspects of news, VeraCity follows a similar 'hybrid' approach by integrating multi-dimensional aspects into a single reliability score to account for both factual content and the style in which it is presented to the reader.

Ghanem et al. (2020) explores the 'emotional fingerprints' of different fake news types. It considers four news types: hoaxes, propagandas, clickbait, and satires. The paper proposes an emotionally-infused, neural network based model (EIN) using Long short-term memory (LSTM). The model relies on word-embeddings and emotional features. Using a separate, purely emotional-features based model, the authors compared it to two baselines: 1. A bag-of-words (BOW) SVM classifier and 2. Word embedding vectors with logistic regression. Their aim through this method was to identify if emotional features independently affect fake news. The research investigates the possibility of using their EIN model as a

clickbait detector, since it was found to provide clear separability of the clickbait class. Their results showed that emotional features alone do clearly detect false news. They also found that detecting suspicious news in Twitter is harder than detecting it in news articles. While their work utilizes complex LSTM neural networks, VeraCity uses slightly different methods like establishing Z-Score benchmarking as a baseline to detect deviations in fake news.

DESIGN REQUIREMENTS

The idea and design of this prototype was inspired by the necessity to help bridge the gap between machine learning & NLP and everyday news consumption without knowing the truthfulness of it. Rather than making definite claims about the integrity level of a text, this prototype uses a multi-metric approach providing more transparency and allowing users to judge the text for themselves.

Functional requirements - These define the outputs of the prototype that must be present for the user.

1. **Multi-Dimensionality Scoring framework**:

   Fake-likelihood score (0 - 100) : To show the factuality of a text by producing a percentage representing the probability of the news being fake. It indicates how closely the input text aligns with known misinformation.

   Bias Score (0 - 100) : To identify presence of personal opinion, subjective language, and lack of factual-neutrality present.

   Content-Reliability Score (0 – 100) : It combines the above two scores and provides an overall 'integrity' check, acting as a final point of judgement for the user.

2. **Tone Interpretation** - Categorize the writing style (tone) of input text based on linguistic intensity, sentiment, and bias. The user-interface will display a brief tone interpretation.
3. **Explainable AI Indicators** - Indicators providing insight into what influenced the model's decision of how fake or real the uploaded text is.
4. **User-Interface** - An interface should be provided for the user to upload their texts easily and be displayed the results in a clear way.

Non-Functional requirements - These define the performance, usability, constraints, and user-experience of the system.

1. **Volume of Processing Capability**: Supports processing of >= 250 words per request considering standard digital news lengths.
2. **Usability and User Satisfaction** - Target User Acceptance Testing (UAT) score of >= 4.0/5.0 through a clean, accessible, and user-friendly interface with a self-explanatory and appealing visual design.

DATASET SELECTION & STRUCTURE (Raw Dataset)

Many datasets were inspected for this project, but most datasets were either narrowly centered on either political content, synthetically generated & lacking real-world news -reporting, or sourced from single platforms. Certain datasets like ISOT and LIAR were disregarded immediately due to their focus on political news only. After exploring more options, some datasets such as "FakeNewsNet" and "Getting Real About Fake News" were also found to be not suitable for use; the former had a small dataset size, while the latter relied on scraped content containing significant amounts of fabricated news respectively.

For building this prototype, the dataset used was "**FineFake: A Knowledge-Enriched Dataset for Fine-Grained Multi-Domain Fake News Detection".** This was chosen specifically because it contained data collected from diverse platforms including both reliable, official news websites and modern social media platforms as sources. It contains data labeled into six distinct topics, proving that it does not focus on only political news. This helps prevent topic bias (data bias) allowing the ML model to train on a variety of data, accurately representing real-world scenarios where data is rarely uniform. Further, each data point in this dataset was reviewed by five professional human annotators using an inter-annotator agreement to eliminate individual bias. The aforementioned points made the FineFake dataset the most suitable and refined choice for this prototype.

The raw, unfiltered dataset was naturally highly-dimensional comprising of **16,909 total samples** and **13 attributes** (columns), namely: **text, image_path, entity_id, topic, fine-grained label, label, knowledge_embedding, description, relation, platform, author, date, comment.** In the unfiltered, unprocessed dataset there were **9407 total samples labeled fake (0)** and **7502 total labeled real (1).** The dataset, even in its raw state, had a more or less inherently balanced class representation with an approximate **fake-labeled to real-labeled ratio** of **56:44.**

The distribution of Real and Fake news by platform was found to be as follows:

PLATFORM	FAKE-LABELED COUNT (0)	REAL-LABELED COUNT (1)
cdc_gov	3	265
apnews	68	666
cnn	237	2073
reddit	3090	958
twitter	591	361
washingtonpost	336	705
snope	5082	2474

TABLE 1 – FAKE AND REAL NEWS DISTRIBUTION BY PLATFORM

METHODOLOGY

Data Cleaning & filtering

Though the dataset provided mostly clean data, with no NaN / null / missing values in the text and label columns, it required some basic 'pre-filter' cleaning before NLP functions could be applied to it. For this, the text column was subject to removal of html tags, correction of merged words (Example: AnimalsTerrestrial → Animals Terrestrial) removal of URLs, removal of emojis and special symbols, and normalization of whitespaces. The cleaning was carried out in exactly the order stated above because URLs and emojis were replaced with whitespaces, hence normalization of whitespaces had to be done at the end. Numbers in the text were not removed, because they are often significant in news texts and represent important information. Here are relevant columns showing a small part of the data before and after the basic cleaning:

	text	clean_text
0	CDCs Abortion Surveillance System FAQsNo, stat...	CDCs Abortion Surveillance System FAQs No, sta...
1	Abortion Surveillance — United States, 2012Abo...	Abortion Surveillance — United States, 2012Abo...
2	RabiesRabies is a fatal but preventable viral ...	Rabies Rabies is a fatal but preventable viral...
3	Other Wild AnimalsTerrestrial Carnivores: Racc...	Other Wild Animals Terrestrial Carnivores: Rac...
4	Coronavirus Disease 2019 (COVID-19)UPDATE\n\nT...	Coronavirus Disease 2019 (COVID-19)UPDATE The ...
5	Venomous Snake Bites: Symptoms & First AidFirs...	Venomous Snake Bites: Symptoms & First Aid Fir...
6	About Zika Virus DiseaseWhat You Need to Know\...	About Zika Virus Disease What You Need to Know...
7	General InformationTriatomine Bug FAQs\n\nWhat...	General Information Triatomine Bug FAQs What i...
8	People at Risk – Pregnant Women and NewbornsCD...	People at Risk – Pregnant Women and Newborns C...
9	Use and Care of MasksConsiderations for specif...	Use and Care of Masks Considerations for speci...

FIGURE 1 – PRE BASIC CLEANING AND POST BASIC CLEANING

After basic data cleaning, a brief analysis of the dataset was carried out in which the word count (the text length) of every data instance was found. Analyzing the text lengths, it was revealed that majority of data samples were in a word-count range of 0-1000. A "minimum density" filter of >= 10 words was applied **post-cleaning.** This was done to ensure it represented only actual semantic content. With this filter, there were a total of **13,862** text samples with **6740 fake** and **7122 labeled real.** This changed the **fake-labeled to real-labeled ratio** to approximately **51:49**, improving the class balance.

Feature Engineering: Sentiment & Linguistic Bias

Following the basic data cleaning, the TextBlob module was used to extract subjectivity and polarity from the basic-cleaned text to capture the author's intent, emotional intensity of the writing, and opinionated claims.

SUBJECTIVITY (BIAS)

Subjectivity quantifies personal opinions and belief within the text. Normally, the style of news reporting is objective and factual-based. By identifying subjectivity (which is in the range 0.0 to 1.0 by default), the prototype was able to use this to provide a "Bias Score" to the users.

POLARITY (EMOTIONAL INTENSITY)

Polarity measures the amount of "emotional intensity" in the text. Sensationalism, a style of news reporting, is marked by the high-volume use of intense (extremely positive or extremely negative) language, most often at the expense of correctness of news being reported. VeraCity quantifies "emotional intensity" in text to determine if the news is written in a way to trigger an emotional response from the reader.

Contrary to traditional sentiment analysis, this analysis was performed on formatted, but pre-normalized text and before applying NLP techniques. While aggressive formatting (excessive HTML tags, chaotic URLs, excessive spacing) was removed, stylistic markers (e.g., exclamation marks, capitalization, etc.), punctuation, and casing were kept to indicate the writing tone and emotional state of the text. They help the prototype preserve the "human aspects" of text pre-vectorization. Here are relevant columns showing a small part of the data with bias and intensity scores:

	text	bias_score	intensity_score
1001	Slacktivism is over. The #NeverAgain movement ...	0.428370	0.111311
1002	These are the victims of the Florida school sh...	0.518495	0.232126
1003	Sandy Hook shooting victims rememberedCNN —\n\...	0.578598	0.268849
1004	Melania Trump hires Cristina Niceta Lloyd Whit...	0.241698	0.096421

FIGURE 2

Linguistic Normalization

To make the text ready for vectorization and ensure that the ml model focused on text keywords only, a spaCy-based pipeline with punctuation removal, stop word removal, lowercasing of text, lemmatization, and tokenization was applied to the clean_text column created in the previous step. Initially, pipeline execution was taking a long time, maybe due to the large dataset size and computational cost of spaCy's dependency parser. To reduce overall execution time, an optimized pipeline with batch processing and batch size of 100 was utilized. This step transformed the text into an nlp_text column consisting of lemmatized, stop-word-filtered tokens. Here is a sample of the relevant columns before and after the nlp pipeline:

	clean_text	nlp_text
12615	A viral meme titled "Choosing Your COVID-19 Va...	viral meme title choose covid-19 vaccine factu...
12616	Asperger's syndrome was named after Hans Asper...	asperger syndrome name hans asperger aid nazi ...
12617	The "entire state of Connecticut" is offering ...	entire state connecticut offer free drink get ...
12618	In the fall of 2021, Lake Superior State Unive...	fall 2021 lake superior state university Issu ...

FIGURE 3

Feature Extraction

I. Vectorization and Word-Embedding Techniques

Many techniques for vectorization exist in NLP. There are frequency-based vectorization techniques such as Bag-of-Words (BoW) and TF-IDF. More sophisticated methods using neural networks, like Word2Vec (implemented using Skip-gram or continuous bag-of-words [CBoW]), also exist. Then there are pre-trained word embeddings, such as GloVe (global vector word embeddings) that create dense vector representations and capture semantic relationships between words. To remain within the scope of this paper, only the method applied in the VeraCity prototype is discussed in detail.

II. TF-IDF

Term Frequency – Inverse Document Frequency is used to measure how important a word (Term) is in a particular document relative to all documents in the corpus. TF-IDF works in two parts. It finds the frequency of each term (TF) relative to total number of words in that specific document. The second part uses inverse document frequency (IDF), the reciprocal of document frequency (DF). DF of a word is the number of documents containing that word. IDF is log-scaled (using natural log) and the product of TF and IDF gives us the formula for TF-IDF:

$$TF - IDF\ (t, d) = tf\,(t, d)\ \times \log\left(\frac{N}{df}\right)$$ **t** = term, **d** = the document, **tf (t, d)** = count of term t in document d, **N** = total number of documents in corpus, **df** = number of documents having term t

TF-IDF captures terms that are frequent in one class, but rare overall. TF increases the score of repeated words within a text, and IDF lowers the score for words common across all texts, so repeated words in fake articles but uncommon elsewhere get high TF-IDF and become distinguishers. It helps identify the word characteristics of certain writing-styles – e.g., identifying recurring buzzwords and emotionally-powered words common in topic-specific or sensational writing style of fake news.

III. Vectorization with TF-IDF

The TF-IDF vectorizer was configured with a maximum of 5000 features and an n-gram range of (1,2) to extract both unique unigrams and bigrams. The vectorizer ranked them by document frequency, kept the top 5000 features, assigned a TF-IDF score to each feature for every row of the text, and created a sparse matrix. The number of features were limited to balance the model detecting the expressiveness of the text and generalization. Since most texts in the filtered dataset had a word count less than 1000, too many features (vocabulary) would have posed an increase in the risk of overfitting.

TF-IDF was chosen over BoW because IDF downplays common, low-significance words, providing more information than raw frequency counts of each word. It was selected over neural word embeddings (Word2Vec / GloVe), because simplicity and interpretability in the prototype were prioritized. TF-IDF allows us to see influential features, required for VeraCity's explainability indicators. Given the dataset size, short length of most texts, and need of fast processing for the user, TF-IDF offered a good balance between performance, transparency, and computational efficiency. Here is a small segment of the high-dimensional sparse matrix generated by the vectorizer:

Note about Figure 4: The snapshot highlights the retention of numerical tokens (e.g., '000', '10 000') alongside linguistic markers.

	000	000 live	000 people	000 vote	000 woman	000 year	01	05	10	10 000	...	young man	young people	youth	zelensky	zelenskyy	zero	zika	zika virus	zone	zuck
0	0.052025	0.0	0.000000	0.0	0.0	0.00000	0.017302	0.0	0.018886	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.000000	0.0	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.000000	0.0	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.054948	0.0	0.042594	0.0	0.0	0.02579	0.000000	0.0	0.029921	0.023822	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.000000	0.0	0.000000	0.0	0.0	0.00000	0.000000	0.0	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 5000 columns

FIGURE 4 – SEGMENT OF TF-IDF FEATURE MATRIX

Model Training and Evaluation

I. SETUP

The final data used for model development included only the vectorized form of the 'nlp_text' column and the 'label' column (where 0 = fake and 1 = real). Five distinct models were trained and tested. These included three variations of Logistic Regression, each with a different regularization strength: logistic regression **(LR**), logistic regression version 2 **(LRV2)**, & logistic regression version 3 **(LRV3),** then a Random Forest classifier **(RFC)**, and a Support Vector Machine **(SVM)**.

The input data was well-balanced, with a **fake : real ratio** of **51:49**. Therefore, majority undersampling and minority oversampling (such as Synthetic minority oversampling technique; SMOTE) methods were not required to be used. Instead, Sklearn's inbuilt class_weight parameter was used. 80% of the data was used for training the models, and 20% for the test set. The split used stratification on the target label to make sure the fake : real ratio was maintained in the training and testing subsets.

Note: To avoid data leakage, the TF-IDF vectorizer was fitted only on the training set. The IDF weights learned are purely from the training samples, imitating a real-world scenario in which entirely unseen texts are encountered.

Model Name	Base Parameters	Distinguishing Parameters of model
LR	Class_weight = 'balanced' max_iter = 1000 random_state = 10	C = 1.0 (default value of C)
LRV2	Class_weight = 'balanced' max_iter = 1000 random_state = 10	C = 10 (weak regularization; low penalty)
LRV3	Class_weight = 'balanced' max_iter = 1000 random_state = 10	C = 0.1 (strong regularization; high penalty)
RFC	Class_weight = 'balanced' random_state = 10	N_estimators = 100 max_depth = None min_samples_split = 10,
SVM	Class_weight = 'balanced' random_state = 10	Kernel = 'linear' Probability = True

TABLE 2 – MODELS-TRAINED SUMMARY

II. HYPERPARAMETER TUNING

A key challenge in high-dimensional text classification is overfitting. However, the first model **(LR)**, showed reasonable results on the training and testing data. To further try and improve this model, regularization was applied and controlled using the hyperparameter ***C***, in Scikit-learn's Logistic Regression. Regularization acts as a "penalty" to discourage the model from assigning an excessively high weight to any single feature, becoming too complex, and over-relying on specific patterns in the training data. This helps it perform better on unseen texts. **C is the inverse of regularization strength ($1/\lambda$)**. Regularization was implemented on the **LR** model in the following ways:

A **high C value** (**10.0 in LRV2**) applied **weak regularization (1/10.0)**. This indicated to the model to prioritize fitting the training data as closely as possible and apply a smaller penalty to the weights. This allowed it to capture complex patterns, but it did risk overfitting on the dataset.

A **low C value (0.1 in LRV3)** applied **strong regularization (1 / 0.1)**. This helped the model prioritize simplicity by keeping the weights small, because it added a heavier penalty to the weights for complexity. This helped the model avoid overfitting and allowed it to generalize better, though it could have led to underfitting, especially if penalty was aggressive.

III. RESULTS & MODEL SELECTION

Model Name	Accuracy	Precision (Fake)	Recall (Fake)	F1-Score (Weighted)
LR	0.7367	0.70	0.80	0.75
LRV2 (C=10.0)	0.70	0.68	0.73	0.70
LRV3 (C=0.1)	0.73	0.68	0.84	0.75
RFC	0.7411	0.70	0.81	0.75
SVM	0.7295	0.69	0.79	0.74

TABLE 3 - MODEL TRAINING RESULTS

Analysis of the results shows a good performance of all the models overall, with the Random Forest Classifier **(RFC)** performing the best with a raw accuracy of **0.7411**. However, further analysis of the performance on the 'Fake' class (Label 0) shows that **LRV3 (Strong Regularization)** performs the best in terms of the **Recall value of 0.84**. This result could be due to the high penalty value of LRV3, improving the generalizability of the model to a larger variety of patterns in fake news. Also, both Logistic Regression **(LR)** and **RFC** have an **F1-score of 0.75**, showing that the TF-IDF vector representation of the text is a strong baseline for classification, regardless of the classification technique applied.

Though the **RFC** has the best accuracy with a difference of **1.1**% from the accuracy of **SVM**, the (linear) SVM was chosen as the final model for the VeraCity prototype. Since the Random Forest model provides the final label as an outcome of vote results from the majority of the trees, the probability (which will directly represent the fake-likelihood score) will be discrete values. On the other hand, the SVM provides a probability which is based on the data point's distance from the **final decision boundary (hyperplane)**.

This is helpful for the prototype, because the **SVM** model provides continuous probability, therefore making it slightly more precise for the fake-likelihood score. Since the model **(SVM, Accuracy: 0.7295, Recall: 0.79)**, is nearly equal to the best model here **(RFC)**, the linear **SVM** was ultimately used finally in displaying the fake-likelihood score on the user-interface.

System Logic & Feature Implementations

I. Fake-Likelihood Score

The core feature of this prototype uses solely the **SVM** model's internal decision to represent the fake-likelihood score. Since a standard binary classification result did not align with the functional requirements of this prototype, the predict_proba() method was used to assign a probability to each of the two classes. The probability of class 0 (fake) was isolated and scaled to a 0-100 range to be displayed as a percentage. Mathematically speaking, this value represents the distance of the user's uploaded text from the hyperplane the model learned based on the training data. The value, essentially, shows the linear **SVM** model's confidence that the text is fake, i.e., belongs to the fake class. The value is not a definite binary classification, rather it falls within a range of 0-100.

II. Bias Score & Z-score Benchmarking

The Bias score for VeraCity was calculated by benchmarking the bias of the user's input text against a statistical average of real (labeled as 1) news from the dataset. As discussed previously in the paper, the subjectivity was extracted for every row (text instance) in the dataset using TextBlob and stored in a new column named as bias_score. All subjectivity scores returned were in the inclusive range (0.0, 1.0) by default. Then, the mean (μ) and standard deviation (σ) of all real instances were calculated and used as a baseline for 'objective' reporting. When a user uploads a text, the system will calculate a bias Z-score for the text. The formulas for this Z-Score calculation are:

$$bias\ difference = subjectivity\ of\ text - Mean\ Bias\ of\ Real\ news$$

$$bias\ Z\_score = \frac{bias\ \ difference}{Standard\ \ Deviation\ \ of\ \ bias\ \ in\ \ real\ \ news}$$

This z-score measures how many standard deviations the input text's bias sits away from the mean of the real-news bias in the training set. To convert this into a user-friendly display score on a scale from 0-100, a multiplier, inspired by the empirical rule in statistics, was applied. Because in a normal distribution (as per this rule), approximately 95% of data falls within two standard deviations of the mean, a z-score of 2.5 was considered as the upper limit to indicate extreme bias. So, any z-score of 2.5 or higher is automatically assigned a bias score of 100 out of 100. For this mapping, each absolute Z-score was scaled by a multiplier of 40. According to this, a standard deviation of 1.25 results in a bias score of 50. This method accurately reflects how much an article's opinionated language deviates from standard bias in real-news reporting.

III. Reliability Score

In a way similar to the bias Z-score calculation, a Z-score for intensity was calculated using polarity values extracted from TextBlob for every instance of the dataset. All intensity values are within the inclusive range (-1.0, 1.0) by default. The mean and standard deviation of intensity for all real news in the training

corpus was found. The Z-score was not scaled by a multiplier this time because this is an internal metric. The average of both un-scaled Z-scores (bias and intensity) is found using the formula:

$$average\ deviation = \frac{Bias\ Z\_Score + Intensity\ Z\_Score}{2}$$

This denotes the average linguistic deviation from real news baselines. The deviation is scaled by a multiplier of 60 to give a final 'linguistic' risk value. A multiplier of 60 means texts that are approximately 1.66 standard deviations away from the normal lead to a 100% risk penalty. The formula used to find the linguistic risk: $risk = average\ deviation \times 60$

To derive the final reliability score, both the **SVM** model's prediction and the linguistic risk are taken into account. To prioritize the ML model's pattern recognition while still accounting for risk associated with highly subjective and emotional text, the model's result is given 80% weightage and the linguistic risk is given 20% weightage in the final reliability score. The formula is as follows:

$$Reliability = 100 - (SVM\ Result \times 0.8) - (Risk \times 0.2)$$

IV. Tone Interpretation

A qualitative context is provided by the prototype in the form of a brief interpretation of the tone. By using a threshold of 0.4 standard deviations, the system is able to identify if a text's linguistic style deviates from the 'neutral' baseline of the training corpus. Depending on where the user's uploaded text falls relative to this threshold, it is grouped into one of the four categories: "highly opinionated and emotional", "subtle opinions and maybe persuasive", "dramatic / sensationalist news", "neutral / standard reporting". These categories are defined based on various combinations of the bias Z-score and the intensity Z-score. Several threshold values were tested (ranging from 0.3 to 0.9), but the threshold of 0.4 was found to be most clearly-distinguishing the real-world test cases into suitable categories. It was used as a common threshold for both bias and intensity. The code showing the logic behind tone categorization:

```
tone_threshold = 0.4
abs_bias_z_score = abs(bias_z_score)
abs_intensity_z_score = abs(intensity_z_score)

if abs_bias_z_score > tone_threshold and abs_intensity_z_score > tone_threshold:
    tone_label = 'Highly Opinionated and Emotional'

# 2. High in Bias OR High in Intensity (Catching the specific leans)

elif abs_bias_z_score > tone_threshold:
    tone_label = 'Subtle Opinions and maybe Persuasive'

elif abs_intensity_z_score > tone_threshold:
    tone_label = 'Dramatic / Sensationalist news'

elif abs_bias_z_score <= tone_threshold and abs_intensity_z_score <= tone_threshold:
        # fewer opinions (less biased) + low emotional language (the IDEAL news standard)
    tone_label = 'Neutral / Standard Reporting'
```

Figure 5 - Tone Interpretation Logic

V. Real and Fake News Indicators / Triggers

The last feature of VeraCity is an explainable AI component (XAI) which displays to the user, certain words that caused the model to classify the text on the fake-likelihood spectrum. A custom extraction logic was

implemented to identify the specific words that influenced the model's decision. First, the nonzero( ) method helped isolate the indices of the specific words that actually appear in the user's current text out of the 5,000-feature vocabulary matrix. Then, the **coefficients (weights)** from the trained, linear SVM were retrieved, where negative values represent "Fake Triggers" and positive values represent "Real Indicators."

By iterating through the active word indices, the code mapped each word to its corresponding learned weight. Finally, these pairs were sorted by their weight values; the five most negative weights were extracted as "Top Fake Triggers," while the five most positive weights were identified as "Top Real Indicators." These indicators and triggers are features learned **during training** that were **also** present in the user-uploaded text, **not** the most important indicators **specifically** from the user's text.

ARCHITECTURAL MODEL OF VeraCity

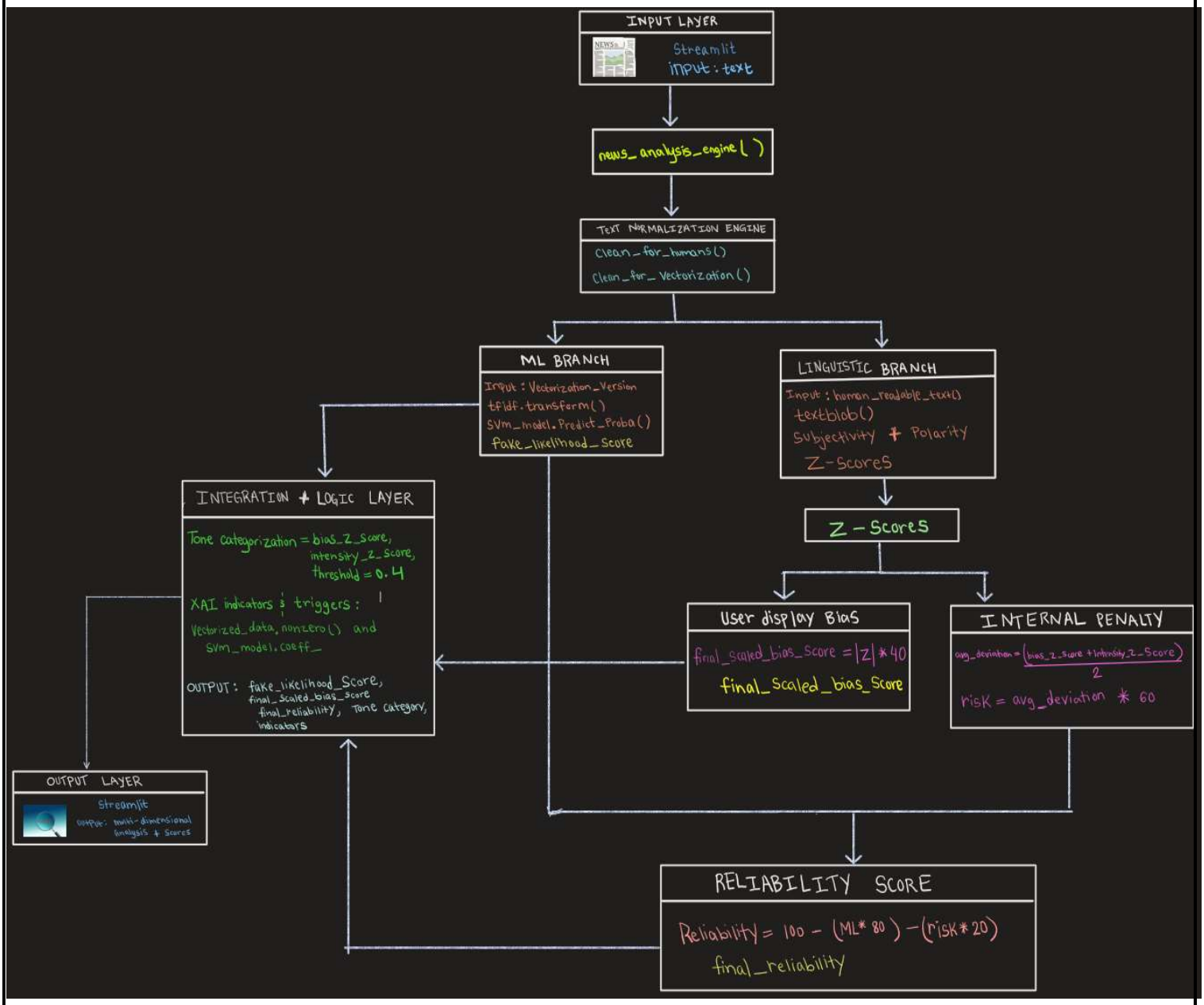

FIGURE 6 - ARCHITECTURAL MODEL OF VERACITY **SOURCE: OWN**

IMPLEMENTATION: SERIALIZATION AND INTERFACE DEVELOPMENT

The Linear SVM itself and the respective TF-IDF transformer were Pickled to save the learned weights and vocabulary. This allowed immediate loading of the trained model to avoid the need for training every time a new text is uploaded. The prototype provides a user-friendly dashboard to present the complex mathematical-based results of the system, like SVM probability and reliability of the text. The frontend of the prototype was built using the Streamlit library. This provided a clean web interface where the user can input text and immediately get real-time results from the system.

OUTPUT AND RESULTS

I. User Interface & Test Setup

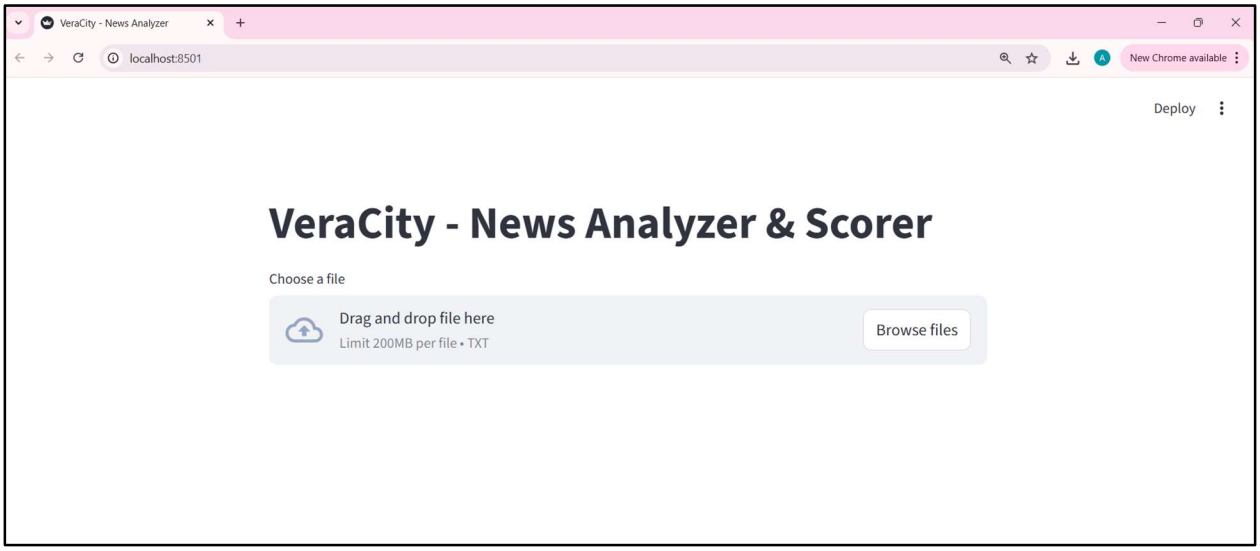

FIGURE 6 - VERACITY WEB APPLICATION VIEW

The final system was tested on **13** real-world texts comprising of real news, fake news, one Facebook post, and satire texts. The fake news consisted of fake medical news, fake conspiracy theories, fake political news, and AI-generated synthetic fake text. The real news consisted of tech-related news, and news about world-issues. Together, all of these were sourced from the following platforms: “BBC”, “Reuters”, “CDC Gov”, “The Onion”, “CNBC”, “Natural News”, “Facebook”, “PolitFact”, “Snopes”, and “Fact Check”.

II. **Demo Case 1 : Fake News**

New research shows just how powerful the phytochemicals in green tea are turning out to be: they are now known to prevent breast cancer, pancreatic cancer, colon cancer, lung cancer and other forms of cancer.

In fact, if green tea were a prescription drug, it would be called a "miracle cancer cure" drug, no doubt. (And it would be sold for $200 a pill, if not more.) But it's not a drug, and it's available to you right now for mere pennies. Green tea is simply one of the most powerful medicinal herbs known. It is especially useful for preventing cancer, and the research keeps on coming.

Every person suffering from cancer (or at risk of being diagnosed with cancer) should be taking green tea nutritional supplements. It has zero negative side effects, and yet delivers powerful anti-cancer

FIGURE 7 - INPUT: FAKE MEDICAL TEXT – 138 WORDS

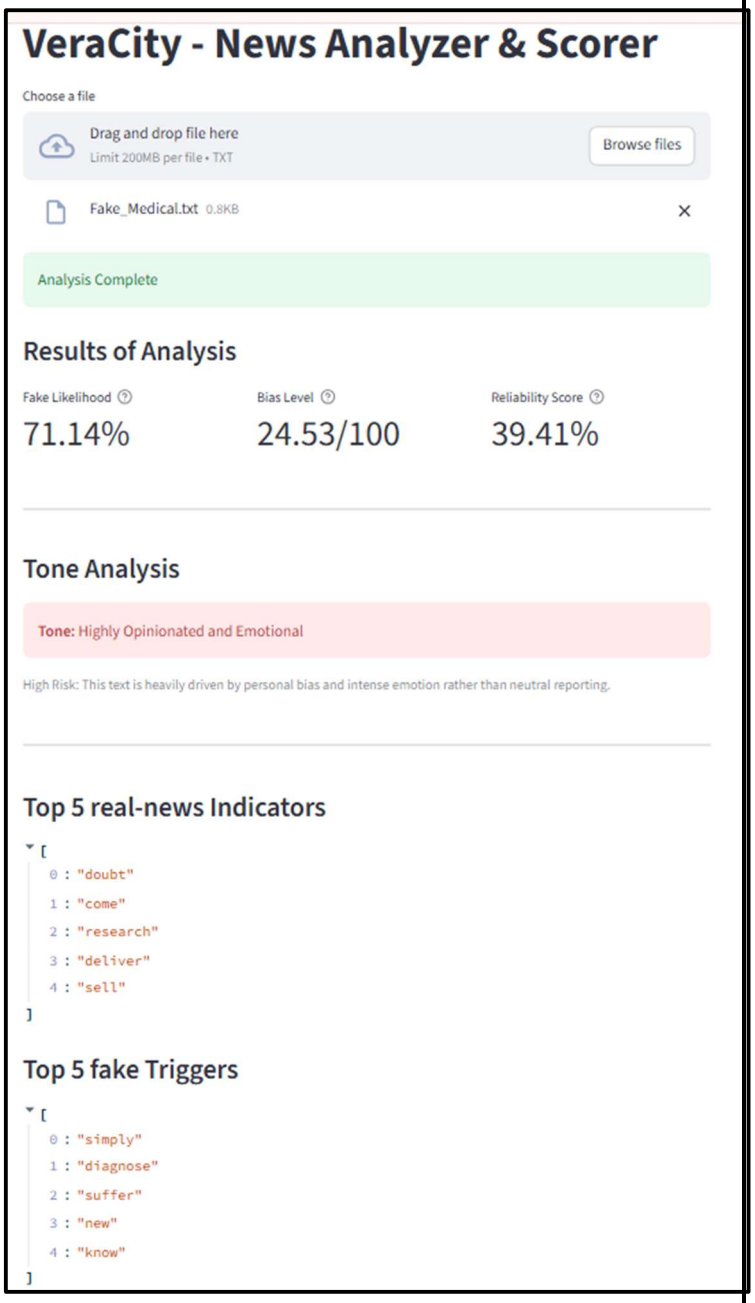

FIGURE 8- DEMO CASE 1 OUTPUT: FAKE NEWS

III. Demo Case 2: Real News

Nvidia boss Jensen Huang on Monday announced Alpamayo, a tech platform the company says will help self-driving cars think like humans.

"Alpamayo brings reasoning to autonomous vehicles, allowing them to think through rare scenarios, drive safely in complex environments, and explain their driving decisions," Huang said on stage at the annual CES technology conference in Las Vegas.

Huang also said Nvidia has begun producing a driverless car powered by its technology, the Mercedes-Benz CLA, in partnership with the German automaker.

The vehicle will be released in the US in the coming months before being rolled out in Europe and Asia. Wearing his trademark black leather jacket, Huang told an audience of hundreds that the project has taught Nvidia "an enormous amount" about how to help partners build robotic systems.

Analysts say the announcement reinforces Nvidia's leadership in integrating AI hardware and software, deepening its push into physical AI.

"NVIDIA's pivot toward AI at scale and AI systems as differentiators will help keep it way ahead of rivals," said Paolo Pescatore, analyst at PP Foresight, from Las Vegas.

"Alpamayo represents a profound shift for NVIDIA, moving from being primarily a compute to a platform provider for physical AI ecosystems."

Shares of the AI chip designer rose slightly in after-hours trading following Huang's presentation.

It featured a video demonstration of the AI-powered Mercedes-Benz driving through San Francisco while a passenger, sat behind the steering wheel, kept their hands in their lap.

"It drives so naturally because it learned directly from human demonstrators," Huang said, "but in every single scenario... it tells you what it's going to do, and it reasons about what it's about to do."

Alpamayo is an open-source AI model, with the underlying code now available on machine learning platform Hugging Face, where autonomous vehicle researchers can access it for free and retrain the model, Huang said.

"Our vision is that someday, every single car, every single truck, will be autonomous," he told the audience.

The company also has plans to launch a robotaxi service by next year in collaboration with a partner, but has declined to name the partner or say where it will be.

Nvidia is the world's most valuable publicly traded company, with a market cap of more than $4.5tn. It became the first company to reach $5tn in October amid the AI boom, but has lost value amid concerns about whether demand for the technology is overhyped.

FIGURE 9 - INPUT: REAL TECH NEWS – 401 WORDS

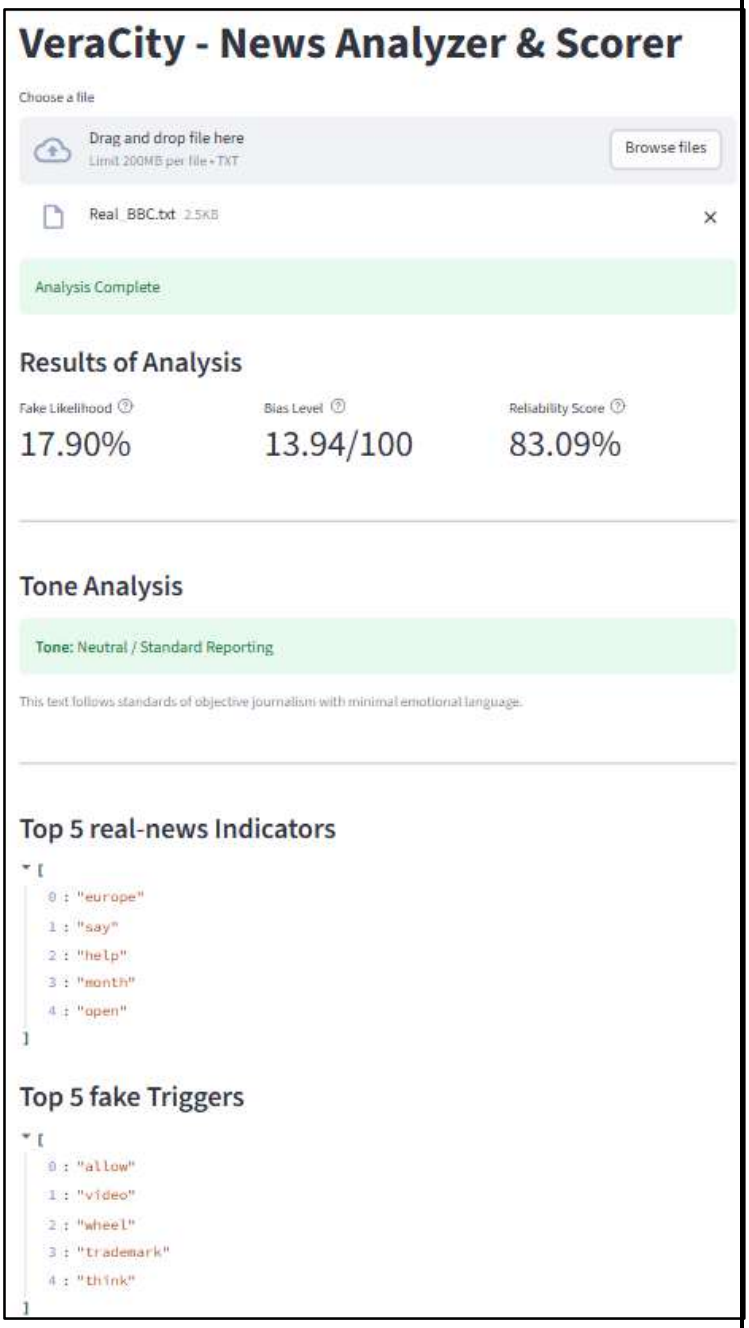

FIGURE 10 - DEMO CASE 2 OUTPUT : REAL NEWS

DISCUSSION

In conclusion, this paper explains the goal, design requirements, methodology, overall process flow, results, and output of the development of the prototype: VeraCity. This system is a multi-metric news analyzer and scorer, with a special focus on detection of fake news. The primary contributions of this prototype include:

1. **A Dual-Path Pipeline:** The development of a hybrid system that combines machine learning with statistical linguistic methods. It makes use of the SVM model's result along with polarity and subjectivity from Python's TextBlob module to incorporate writing style, tone, and use of emotional-language in providing a holistic and well-rounded overview of the overall reliability and integrity of a news text. This is a system that goes beyond binary classification of news.

2. **Explainable Outputs on User-Interface:** The integration of triggers of fake news and indicators of real news is an attempt to address an opaque and sometimes 'black-box' like nature of machine learning systems. Clear visual feedback of influential terms in the SVM model's classification gives users a glimpse into the model's working and understanding of the text.

3. **Rule-Based Benchmarking Framework:** This system uses statistical-based heuristics which included benchmarking against real news from the training corpus. The thresholds and quantitative metrics were decided carefully and the logic of this system uses real news as a baseline for comparisons.

Limitations

Despite the system's high accuracy in some test cases, several limitations were identified during testing:

1. **Semantic Depth:** By using **TF-IDF** instead of Word Embeddings (like Word2Vec or BERT), the model focused on word frequency and importance, as expected, but ignored deep semantic context. This did result in this system being unable to understand the meaning, relationships between words, and the context.

2. **Performance drop in Domain-Specific Text:** While the model performed well on one of the fake medical news samples in the demo, it had difficulty detecting a different one, also a fake medical news text. This shows presence of inconsistency in the model. This suggests that without domain-specific training, the model struggles with complicated medical terminology and domain-specific words.

3. **Satire Detection:** The model also currently struggles to distinguish between fake news and **satire**. Since Satire content is intentionally written in a way to deceive the reader, the model was unable to give a reasonable fake-likelihood score for both the fake satire texts tested. However, the tone categorization was justified for the two satire texts.

4. **Text Length Paradox:** Surprisingly, the model performed better at detecting misinformation in **shorter texts**. This is likely because of the "dense" nature of news headlines, where every word is written to maximize its impact on the reader, whereas longer fake articles caused the model to overlook the stylistic indicators of fake news because of large amounts of vocabulary.

Lessons Learned

1. **Fake and Real news classification is a vast concept**: A major takeaway from this project was that there are countless features which can and should possibly be considered in classifying or detecting fake news. Not just a text's content and tone or writing style can be used to develop a truly authentic system. The text's source credibility, social media engagement of the news, and countless other metadata and features are needed. Also, since information changes and propagates rapidly, this task is even more difficult as it might require some form of continuous updates to the system's data and logic.

2. **The "Headlining" Effect**: I learned that misinformation is often concentrated in shorter snippets of text. While I expected long articles to provide more information for the SVM to perform better, some of the most successful detections actually occurred on shorter texts where the deceptive intent was forced into a few emotional phrases or sentences. A few news headlines (solely the headlines) were tested and the system was able to detect their falseness with more confidence. Although the tests of these headlines were not included in the project due to the 250-word rule processing design requirement, I learned that a system like this may work better in an environment with shorter texts, such as Instagram posts or short Twitter posts.

3. **The Limitations of Frequency-Based Vectorization:** Using TF-IDF taught me that while counting words is computationally efficient for a fast web app and provides transparency for XAI components, it lacks "memory." I realized that without semantic embeddings, the model treats related features (in terms of meaning) as entirely different concepts, which could be a significant hurdle for future-proofing detection.

Future work

Though the system currently functions to meet all the design requirements and goals of this project, it still has certain limitations as explained above and room for improvement. Here are some of the following ways in which this prototype can be made better:

1. In this system, TF-IDF was chosen because of simplicity and need for indicators, though implementing Word2Vec or transformers to capture semantic meaning could help the model in correctly identifying the falseness of a larger variety of texts than it is currently able to.

2. Implementing a multi-modal analysis where metadata (e.g., author, source) and perhaps associated images are also used to detect fake news or maybe mismatched headlines. This would be better implemented on news posted on social media platforms, like Reddit and Twitter, where there is sometimes an image accompanying the text.

3. Developing a model / classifier specifically for identifying Satirical news texts, or a system consisting of multiple models, each trained specially to detect a certain type of fake news.

References

1. Coleman, A. (2020, August 12). *'Hundreds dead' because of Covid-19 misinformation*. Retrieved January 2026, from BBC: https://www.bbc.com/news/world-53755067

2. Sardarizadeh, E. T. (2024). How a deleted LinkedIn post was weaponised and seen by millions before the Southport riot. BBC.

3. Salve, R. R. (2025). *Fake News Detection: Leveraging Natural Language Processing and Machine Learning for Reliable Information Verification*. Rochester Institute of Technology. https://repository.rit.edu/cgi/viewcontent.cgi?article=13302&context=theses

4. Kuntur, S., WrĂłblewska, A., Paprzycki, M., & Ganzha, M. (2024). Fake News Detection: It's All in the Data!. *arXiv preprint arXiv:2407.02122* https://arxiv.org/html/2407.02122v2

5. Oriola, O. (2021). Exploring N-gram, word embedding and topic models for content-based fake news detection in FakeNewsNet evaluation. *Int. J. Comput. Appl*, *975*, 8887. https://www.ijcaonline.org/archives/volume176/number39/oriola-2020-ijca-920503.pdf

6. Hamed, S. K., Ab Aziz, M. J., & Yaakub, M. R. (2023). A review of fake news detection approaches: A critical analysis of relevant studies and highlighting key challenges associated with the dataset, feature representation, and data fusion. *Heliyon*, *9*(10). https://pmc.ncbi.nlm.nih.gov/articles/PMC10539669/

7. Emil, R. Ș., & Remus, B. (2025). A Review of Automatic Fake News Detection: From Traditional Methods to Large Language Models. *Future Internet*, *17*(10), 435. https://www.mdpi.com/1999-5903/17/10/435

8. Pérez-Rosas, V., Kleinberg, B., Lefevre, A., & Mihalcea, R. (2018, August). Automatic detection of fake news. In *Proceedings of the 27th international conference on computational linguistics* (pp. 3391-3401). https://aclanthology.org/C18-1287/

9. Shu, K., Mahudeswaran, D., Wang, S., Lee, D., & Liu, H. (2020). Fakenewsnet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media. *Big data*, *8*(3), 171-188. https://www.liebertpub.com/doi/abs/10.1089/big.2020.0062

10. Ghanem, B., Rosso, P., & Rangel, F. (2020). An emotional analysis of false information in social media and news articles. *ACM Transactions on Internet Technology (TOIT)*, *20*(2), 1-18. https://dl.acm.org/doi/abs/10.1145/3381750

11. Gandhi, S. S. (2023). *Research paper recommendation system using natural language processing* [Bachelor's thesis, Vellore Institute of Technology]. mediatum. Technical University of Munich. https://mediatum.ub.tum.de/doc/1713430/48ua6wiwivmsgraeeq2bn4stp.pdf

12. Zhou, Z., Zhang, X., Zhang, L., Liu, J., Wang, S., Liu, Z., ... & Yu, P. S. (2024). Finefake: A knowledge-enriched dataset for fine-grained multi-domain fake news detection. *arXiv preprint arXiv:2404.01336*. https://arxiv.org/abs/2404.01336

13. https://github.com/Accuser907/FineFake

Appendix

```
import streamlit as st
from VeraCity_engine import *
import pickle

# Unpickle file - loading its contents
# use streamlit's decorator st.cache_resource to prevent reloading all contents of pickle file everytime
# upon every small change (uploading a file, change in code) streamlit re-runs the entire script of python, so everytime a file is
uploaded it will rerun the entire main.py
# st.cache_resource prevents that, once this is loaded, saves it, and does NOT rerun this part every time

@st.cache_resource
def load_veracity_bundle():
  with open('VeraCity_model.pkl', 'rb') as file:
    return pickle.load(file)

veracity_bundle = load_veracity_bundle()

st.set_page_config( page_title = 'VeraCity - News Analyzer')

st.title("VeraCity - News Analyzer & Scorer")

uploaded_file = st.file_uploader('Choose a file', type = "txt")

if uploaded_file is not None:
  # Convert uploaded file to string
  article_text = uploaded_file.read().decode('UTF-8')

  with st.spinner('Analyzing...'):
    fake_likelihood, bias_score, reliability, tone_label, top_fake_triggers, top_real_indicators =
news_analysis_engine(article_text, veracity_bundle)
    # Process the text by passing to news_analysis_engine function; also pass the pickle file that was just loaded
    st.success('Analysis Complete')

# ------------------------------------------------------------------------------------------
  # displaying results
  st.subheader('Results of Analysis\n')
  col1,col2,col3 = st.columns(3)

  # 1. ML PREDICTION
  col1.metric(
      label="Fake Likelihood",
      value=f"{fake_likelihood:.2f}%",
      help="Probability that the content matches known misinformation patterns."
  )

  # 2. LINGUISTIC CONTEXT - BIAS & TONE
  col2.metric(
      label="Bias Level",
      value=f"{bias_score:.2f}/100",
      help="Measures subjective and emotional language compared to Real news benchmarks."
  )

  # 3. RELIABILITY
  col3.metric(
      label="Reliability Score",
      value=f"{reliability:.2f}%",
      help="The overall confidence rating of the article's integrity.")


  # Logical Color Mapping
  st.markdown("---")
  st.subheader("Tone Analysis")

  # Mapping your Step 10 & 11 logic to UI visuals
  if tone_label == 'Neutral / Standard Reporting':
      st.success(f"**Tone:** {tone_label}")
      st.caption("This text follows standards of objective journalism with minimal emotional language.")

  elif tone_label == 'Subtle Opinions and maybe Persuasive':
      st.info(f"**Tone:** {tone_label}")
      st.caption("The text includes personal viewpoints or persuasive words, but language remains calm and professional.")

  elif tone_label == 'Dramatic / Sensationalist news':
```

```
        st.warning(f"**Tone:** {tone_label}")
        st.caption("Warning: This article uses loud, emotional, or exaggerated language designed to provoke a reaction.")

    elif tone_label == 'Highly Opinionated and Emotional':
        st.error(f"**Tone:** {tone_label}")
        st.caption("High Risk: This text is heavily driven by personal bias and intense emotion rather than neutral reporting.")

    st.markdown("---")
    st.subheader("Top 5 real-news Indicators")
    st.write(top_real_indicators)

    st.subheader("Top 5 fake Triggers")
    st.write(top_fake_triggers)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from bs4 import BeautifulSoup
from textblob import TextBlob
import spacy
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

nlp = spacy.load('en_core_web_sm')

# --- 2. CLEANING FUNCTION ---

def clean_text_for_length(text):
    """
    Applies basic cleaning steps (HTML, URLs, Emojis, Whitespace) to the text column; which is treated as a Series here.
    Light cleaning for exploratory analysis and getting text length statistics.
    Does NOT perform any normalization of text like lemmatization, stopword removal, lowercase, remove punctuations.
    """

    # 1. Remove HTML tags - using the BeautifulSoup library and its built in HTML parsing engine

    def remove_html_tags(text):
        if pd.isna(text) or text is None:
            return text

        cleaned_html = BeautifulSoup(text, 'html.parser').get_text()

        # Fixes the merged words (e.g., "AnimalsTerrestrial" -> "Animals Terrestrial")
        cleaned_html = re.sub(r'([a-z])([A-Z])', r'\1 \2', cleaned_html)

        return cleaned_html

    # 2. Remove URLs - Using same regex pattern as in the count_url function above.
    url_pattern = r'https?://\S+|www\.\S+|\S+\.\S+/\S+'

    # 3. Remove Emojis / symbols
    emoji_pattern = re.compile(
        '['
        '\U0001F600-\U0001F64F'  # emoticons
        '\U0001F300-\U0001F5FF'  # symbols & pictographs
        '\U0001F680-\U0001F6FF'  # transport & map symbols

        '\U0001F700-\U0001F77F'  # alchemical symbols
        '\U0001F900-\U0001F9FF'  # supplemental symbols and pictographs
        ']+',
        flags = re.UNICODE
    )

    # Applying cleaning sequentially
    cleaned_text = text.astype(str).apply(remove_html_tags)

    cleaned_text = cleaned_text.str.replace(url_pattern, ' ', regex = True)
```

```
    cleaned_text = cleaned_text.str.replace(emoji_pattern, ' ', regex = True)

    # 4. Normalize Whitespace
    cleaned_text = cleaned_text.str.strip()
    cleaned_text = cleaned_text.str.replace(r'\s+', ' ', regex=True)

    return cleaned_text


def get_bias(text):
    """
    INPUT: text (row-wise) from usable_data DF
    OUTPUT: bias of text
    """
    text = str(text)
    text = TextBlob(text)
    bias = text.sentiment.subjectivity
    return bias

def get_intensity(text):
    """
    INPUT: text (row-wise) from usable_data DF
    OUTPUT: polarity of text (how negative or positive the tone is)
    """
    text = str(text)
    text = TextBlob(text)
    polarity = text.sentiment.polarity
    return abs(polarity)
    # we take the absolute value because extremely positive (1.0) and extremely negative (-1.0) both are high intensity


def nlp_pipeline_optimized(texts_series):
    """
    INPUT: pandas series; clean_text

    Purpose of this function:
    - Lowercases everything
    - Tokenizes (split words)
    - Lemmatizes (finding word roots)
    - Removes Stopwords (common, but not useful words: the, is, at)
    - Removes Punctuation (but KEEPS numbers; numbers are often significant in news texts)

    OUTPUT: list of cleaned strings
    """

    processed_texts = []

    # Adding a limit of 5000 words.
    texts_to_process = texts_series.astype(str).apply(lambda x: " ".join(x.split()[:5000])).str.lower()

    # nlp.pipe takes an ITERABLE (the whole column)
    # batch_size = 100 means it processes 100 articles (100 rows) at a time in memory

    docs = nlp.pipe(texts_to_process, batch_size = 100)                                    # Creates a doc generator; use nlp.pipe for
batch processing, optimization, and to speed up
                                                                                           # nlp.pipe() can take a pandas Series as
input because a Series is an iterable of strings


    for doc in docs:                                                                      # Creating the doc object from generator for
each row at a time(this tokenizes the text and stores the tokens)
        tokens = [token.lemma_.lower() for token in doc if not token.is_stop and not token.is_punct and not token.is_space]

        # newlines and tabs are treated as spaces, so we must explicitly use 'not token.is_space' eventhough spacy does not
generally consider spaces as separaate tokens.
        # Extracts root of the word (the lemma) IF it's not a stopword or punctuation or a white space

        processed_texts.append(" ".join(tokens))

    return processed_texts

# NOTE: a doc object is created for every row separately still, but now 100 rows are processed parallely.
# so saying 'for doc in docs' still gives us one article at a time.

def clean_for_humans(text):
    # At this stage, I want to keep punctuation and capitalization, for the bias and intensity calculations
    # Using the clean_text_for_length function used on the FineFake dataset above
```

```
    # Removes HTML, URLs, and Emojis but KEEPS casing and punctuation;
    # Also spaces out two capitalized words with no space (E.G.: AnimalsTerrestrial)
    text_series = pd.Series([text])
    cleaned = clean_text_for_length(text_series)
    return cleaned[0]

def clean_for_vectorization(text):
    # Running the full nlp_pipeline_optimized
    # Lemmatizes and removes stopwords so TF-IDF can read it
    text_series = pd.Series([text])
    nlp_text_list = nlp_pipeline_optimized(text_series)
    return nlp_text_list[0]

def news_analysis_engine(user_input, bundle):

    # FAKE-LIKELIHOOD SCORE

    # STEP 1: Basic cleaning (HTML/URLs) - this version is for tone / bias / LLM
    human_readable_text = clean_for_humans(user_input)

    # STEP 2: Deep NLP cleaning- this version is ONLY for the ML Model; the version which will be vectorized using tf-idf
    vectorization_version = clean_for_vectorization(human_readable_text)

    # ------------------------------------------------------------------------------------------------------------------------------------------
    # STEP 3: Fake News Likelihood
    vectorized_data = bundle['vectorizer'].transform([vectorization_version])
    prediction_prob = bundle['model'].predict_proba(vectorized_data)        # Returns a 2D numpy array (n_samples, n_classes)
    fake_likelihood_score = prediction_prob[0][0] * 100              # Selects row 0 (the first sample) and column 0 (probability of class 0 = Fake)

    # ------------------------------------------------------------------------------------------------------------------------------------------

    # BIAS SCORE

    # STEP 4: Bias Score (Using Human Version)
    blob = TextBlob(human_readable_text)
    current_subjectivity = blob.sentiment.subjectivity

    # STEP 5: Calculating the Z-Score (The distance from mean of real news; NOTE: the mean and std from the FineFake REAL (1) news (training + testing)
    # is used to serve as a benchmark; to keep comparisons fair

    bias_diff = current_subjectivity - bundle['REAL_BIAS_MEAN']        # REAL_BIAS_MEAN = mean of bias of ALL real-labeled texts in filtered dataset
    bias_z_score = bias_diff / bundle['REAL_BIAS_STD']                 # bias_z_score = how many standard deviations away is text's subjectivity from real-news subjectivity.



    # STEP 6: Converting the bias_z_score to a percentage (scale 0 - 100)

    # NOTE: 1.25 STD away (either in positive or negative direction) is considered to be 50% biased using 40 as the multiplier.
    # So 2.5 STDs away from the mean is 100% biased.
    # By using a multiplier of 40:
    # 1.0 STD away = 40% Bias Score
    # 2.0 STD away = 80% Bias Score
    # 2.5 STD away = 100% Bias Score (Top 1% of outliers)
    # This is inspired from the empirical rule of standard deviation, which states that roughly 68% of data is WITHIN the range of 1 STD from the mean.
    # So if an article is 1 STD away from the mean here (NOT WITHIN 1 STD, but >= 1 STD), it is moving away from the 'most common' zone, where most data lies.
    # The rule also states that 95% of all data should fall within 2 STDs from the mean.
    # So if a text is exactly 2.5 (or more) STDs away from the mean [I.E., further away than 95% of the texts from the mean], it should get a bias score of 100%.
    # So 1.25 STD = 50% bias score and 2.5 STDs = 100% bias score; this is the conversion method used to convert from Z-score to a bias scale 0-100


    final_scaled_bias_score = abs(bias_z_score) * 40
    if final_scaled_bias_score > 100:
        final_scaled_bias_score = 100
    # In case score exceeds 100 when converting to a scale of (0-100), consider max always: 100
    # this is the user-display scaled bias score
```

```
    # ------------------------------------------------------------------------------------------------------------------------------------------
--------------------



    # POLARITY

    # STEP 7: Getting Polarity (Intensity) - use 'abs' because extreme +/- both show high emotional language
    # Intensity is the strength of the emotion
    current_polarity = blob.sentiment.polarity
    current_intensity = abs(current_polarity)

    #  STEP 8: Calculating Intensity Z-Score
    intensity_diff = current_intensity - bundle['REAL_INTENSITY_MEAN']
    intensity_z_score = intensity_diff / bundle['REAL_INTENSITY_STD']

    # ------------------------------------------------------------------------------------------------------------------------------------------
------------------------



    # AVERAGING BIAS AND POLARITY

    # STEP 9: Linguistic Score
    # Creating a combined deviation score (Bias + Intensity)
    # Bias and intensity are both taken into account by taking their average.
    avg_deviation = (max(0, bias_z_score) + max(0, intensity_z_score)) / 2



    # STEP 10: Scaling the deviation to a 0-100 penalty score.
    # a higher multiplier (60) for the internal risk calculation than the display score (40) to be more aggressive against
misinformation.
    # 1.0 STD deviation = 60 point penalty (internal)
    # 1.66 STD deviation = 100 point penalty (internal)
    risk = min(100, avg_deviation * 60)

    # ------------------------------------------------------------------------------------------------------------------------------------------
-------------------



    # STEP 11:

    # TONE EXPLANATION

    # Both the bias (subjectivity) and Intensity (Polarity) are considerd to categorize a text to a tone.
    # the bias and intensity Z-Scores are used in combination
    # bias z-score tells us how opinionated the text is and intensity z-score tells us how loud / emotional language is
    # compared TO AVERGAE POLARITY OF REAL-LABELED NEWS seen in the FineFake dataset

    # Using 0.4 as the threshold; any text more than 0.4 Standard Deviations away from the mean
    # is flagged as "moving away from neutral" to ensure even subtle bias is caught.

    tone_threshold = 0.4
    abs_bias_z_score = abs(bias_z_score)
    abs_intensity_z_score = abs(intensity_z_score)

    if abs_bias_z_score > tone_threshold and abs_intensity_z_score > tone_threshold:
        tone_label = 'Highly Opinionated and Emotional'

    # 2. High in Bias OR High in Intensity (Catching the specific leans)

    elif abs_bias_z_score > tone_threshold:
        tone_label = 'Subtle Opinions and maybe Persuasive'

    elif abs_intensity_z_score > tone_threshold:
        tone_label = 'Dramatic / Sensationalist news'

    elif abs_bias_z_score <= tone_threshold and abs_intensity_z_score <= tone_threshold:
            # fewer opinions (less biased) + low emotional language (the IDEAL news standard)
        tone_label = 'Neutral / Standard Reporting'



    # ------------------------------------------------------------------------------------------------------------------------------------------
--------------------------
    # STEP 12:

    # RELIABILITY

    # STEP 11:  Reliability
    # Note: This is actually a content-based reliability score, but it can be used to derive conclusions about the source.
```

```
    # That interpretation is left upto the user.
    # Simple formula: 100 minus weighted penalties. Penalty 'points' are deducted from 100 to get this score.
    # 2 penalties:
                # 1. fact / bias check using fake likelihood score from ml model            (80% score from ML model)
                # 2. tone / style check using risk which combines bias and intensity        (20% from risk = Bias + Intensity
combined)


    # tone check (risk) has lower weightage (20%) because a text can be a bit biased but still be factually true.
    # Bias and fake-likelihood score should not be interpreted together or based off each other.
    # True texts can have some bias, as it depends on the author and their writing-style.

    fact_penalty = fake_likelihood_score * 0.80
    risk_penalty = risk * 0.20

    final_reliability = 100 - fact_penalty - risk_penalty
    if final_reliability < 0:
        final_reliability = 0                   # To ensure score does NOT drop below 0 ever.
    if final_reliability > 100:
        final_reliability = 100                 # To ensure score does NOT cross 0 ever.

    # ------------------------------------------------------------------------------------------------------------------------
--------------------------

    # STEP 13:

    # TOP-CONTRIBUTING WORDS

    # Get indices all words from array (created during vectorization) that appear in new text uploaded by user
    # Returns a tuple of (nonzero_row_index, nonzero_col_index)
    # Example: That is (array([0, 0, 1, 1, 2]), array([0, 2, 1, 2, 0]))
    # Non-zero is present at (0,0), (0,2)....etc.
    # here, the rows will be indices of texts from each row and columns will be the unique words / bigrams
    # SO rows not needed (first element of returned tuple not needed)
    # Since you're only uploading one text at a time for prediction, your vectorized_data has shape (1, n_features).
    # That means there's only one row (row 0).
    # The first element of the tuple (row_indices) will just be [0, 0, 0, …].
    # The second element (col_indices) is the one that matters: it tells you which words from the vocabulary actually appear in
this uploaded text.

    nonzero_indices = vectorized_data.nonzero()[1]
    feature_names = bundle['vectorizer'].get_feature_names_out()

    # Get SVM weights (coefficients) of linear
    # one weight assigned to each tfidf feature (each word / bigram chosen by vectorizer)
    # Positive weights = Real indicators, Negative weights = Fake indicators
    # negative represents one class, positive the other (for binary classification)
    weights = bundle['model'].coef_.toarray().flatten()

    scored_words = []
    for index in nonzero_indices:
        word = feature_names[index]
        weight = weights[index]
        scored_words.append((word, weight))

    # Sort: Lowest weights (Fake triggers) first, Highest weights (Real markers) last
    scored_words.sort(key= lambda x: x[1])

    top_fake_triggers = [w[0] for w in scored_words[:5]]    # lowest numbers (negative) imply fake label
    top_real_indicators = [w[0] for w in scored_words[-5:]]    # higher numbers (positive) imply real label
    # Selects first element of tuple (the word itself) for first 5 (most negative weights)
    # Selcts first element of tuple (word itself) for last 5 tuples (ones with most positive weights)

    # Use SVM weights, because it helps us see which words pushed the decision towards fake or real (in this case, the
probability) the most

    return fake_likelihood_score, final_scaled_bias_score, final_reliability, tone_label, top_fake_triggers, top_real_indicators
```