AI-Generated Text Classifier

Taiyo Williamson, Ezequiel Beck
CS 463/663 – Foundations of Machine Learning (Spring 2025)

May 16, 2025

Code: `https://github.com/taiyyoson/ml_text_classifier_project`

Abstract

The problem we tried to address through this machine learning project is trying to distinguish and classify AI-generated text from human text. After EDA and feature engineering in which we found some key distinguishing factor between the type of texts, we managed to get some good accuracy results from the models we implemented. Although this project is just starting out and we continue developing working, it is significant as it gives us insight into how we can further distinguish AI from human text and provide more clarity on how to solve this problem.

Contents

1 Introduction

One of the main reasons we chose to go with this project was due to the growing relevance and presence of artificial intelligence (AI) and AI-generated content over the last few years. The rise of tools like ChatGPT, Midjourney, and other generative AI platforms has changed how people interact with technology on a daily basis. We have all witnessed a dramatic increase in the capabilities and availability of AI tools. It feels like every week there's a new model, a new breakthrough, or an innovative product being released that pushes the boundary of what we thought was possible just a year ago. This incredible pace of development is both exciting and, at times, a bit overwhelming. For that reason, we were extremely interested in taking a step back and analyzing one of the more controversial effects of these tools: the use of AI to generate written content.

Because we are part of an academic institution, this issue felt especially relevant given how AI is being used in-class. The rise of AI-generated writing has created a lot of discussion among students, professors, and administrators alike. There's a growing concern about how to maintain academic integrity in an age where essays, assignments, and discussion posts can be generated with just a few prompts. This naturally led us to the idea of building a model that could help detect whether a given piece of text was written by a human or generated by an AI. From both an ethical and technical standpoint, this seemed like a very interesting and challenging project.

The audience that would benefit from a project like this primarily includes those in the education sector. Professors, teaching assistants, and even admissions committees could potentially use a tool like this to flag questionable content and better understand how students are using (or misusing) AI tools. But beyond education, this problem has broader implications. In journalism, publishing, and online media, being able to distinguish between authentic human writing and machine-generated content could play a big role in combating misinformation and preserving credibility. Another type of audience this could be affect could actually be recruiters. There often are a lot of questions on job applications which people applying could easily answer with the result of a prompt. Recruiters could use projects like this to help them weed out lazy people applying to the jobs.

As you go through this report, you'll see how we approached this task—from choosing a dataset to conducting exploratory data analysis (EDA), engineering features, training models, and interpreting results. Our findings may be a bit surprising or confusing in places, but they reflect the complexity of the problem we're dealing with. Not only is distinguishing AI from human generated text hard but applying models like BERT is harder than it looks. That difficulty could be a reason to why our BERT model is getting results that we weren't expecting. Detecting AI-generated text is not as simple as it might seem, and we hope our exploration of this challenge gives you insight into both the technical side and the broader context of working with modern AI systems.

2 Methods

2.1 Dataset(s) & Preprocessing

We worked with two datasets in this project. The primary dataset was a labeled CSV file of essays, with each sample marked as either human-written or AI-generated. Additionally, we used a large-scale public dataset, `artem9k/ai-text-detection-pile`, accessed via the Hugging Face `datasets` library. This dataset contains over 1.3 million examples and provides broader coverage of generative models and writing sources.

Variables. Each dataset includes:

- `text` – The raw text of each essay or passage.
- `generated` – Binary label indicating authorship (0 = human, 1 = AI).

In the Hugging Face dataset, the `source` column was mapped to `generated` and cleaned by removing unused columns like `id`.

Data Access. The Hugging Face dataset was accessed through Python's `datasets` API. The CSV was read locally using `pandas` and shuffled for randomness using `sample(frac=1)`.

Preprocessing. Several cleaning and transformation steps were applied:

- **Shuffling and index reset** to avoid ordering bias.
- **Label encoding** for categorical fields.
- **Missing value check** using `isnull()` confirmed no major gaps.
- **Dropping unnecessary columns** such as `id` and unused metadata.
- **TF-IDF vectorization** on the `text` field with a cap of 10,000 features.
- **Combining numeric and vectorized features** using sparse matrix stacking.
- **Train-test split** using an 80/20 ratio with `train_test_split`.

These preprocessing steps ensured a clean, structured dataset suitable for downstream modeling with logistic regression and random forest classifiers.

2.2 EDA & Feature Engineering

To enhance our model's ability to distinguish between AI-generated and human-written text, we engineered several structural and readability-based features. The goal of these features was to to capture patterns in the text which could help us find out what makes it differ across the two classes.

Engineered Features. We made the following features from the raw text:

- `text_length`: Total number of characters in each essay.
- `word_count`: Number of words, computed via tokenization.
- **Readability Metrics:**
  - **Flesch Reading Ease** (`readability`) – Higher values = easier to read.
  - **Gunning Fog Index** (`gunning_fog`) – Estimates education level required; clipped to [5, 20].
  - **SMOG Index** (`smog_index`) – Alternative complexity metric, effective for short texts.

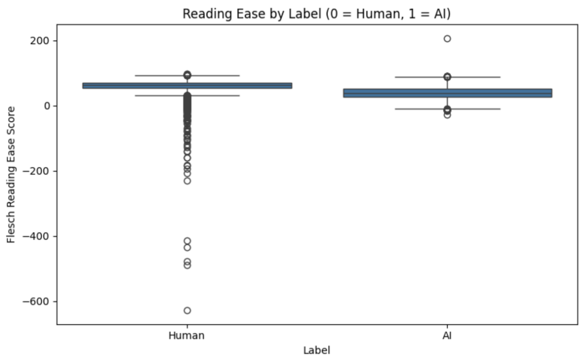


Figure 1: Flesch Readability Scores

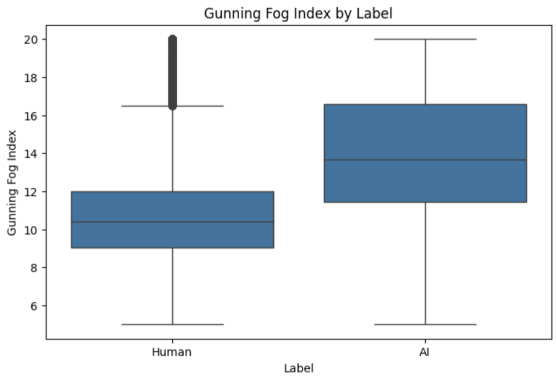


Figure 2: Gunning Fog Scores

These features were chosen to reflect structural and linguistic traits that may differ between machine- and human-written content.

Summary Statistics. Exploratory analysis showed measurable differences in text length, word count, and readability across the two classes. For example, AI-generated texts often had different distributions in character length and complexity scores. Visualizations such as histograms and boxplots were used to illustrate these trends.

Data Quality. We confirmed that the dataset was clean and suitable for analysis:

- No significant missing values were found.
- All feature data types were appropriate.
- Gunning Fog values were clipped to reduce the influence of outliers.
- The dataset size was sufficient for training reliable models, with additional experimentation on a larger variant of over 1.3 million entries.

2.3 Models

In this section, we describe in detail the machine learning models implemented for the classification task of AI-generated versus human-written text. We selected models with complementary strengths—Logistic Regression for its interpretability and efficiency, and Random Forest for its robustness and ability to capture nonlinear patterns. Each model was trained on engineered linguistic features alongside TF-IDF representations of the text data.

2.3.1 Model 1: Logistic Regression

Logistic Regression is a linear model commonly used for binary classification problems (as compared to Linear, Polynomial, LASSO, and Ridge, which are for regression problems on continuous data). Estimates the probability that a given input belongs to a particular class using the sigmoid function:

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x} + b)}}$$

where $\mathbf{x}$ is the feature vector, $\mathbf{w}$ is the weight vector, and b is the bias term. The model is trained to minimize the log-loss (binary cross-entropy), which encourages confident and correct predictions.

This model is well-suited for our task because:

- It performs well with high-dimensional sparse data (like TF-IDF vectors).
- It is interpretable—coefficients reveal which features are most influential.
- It is relatively fast to train, making it ideal for rapid experimentation.

Hyperparameters used:

- `C` (inverse regularization strength): Controls the amount of L2 regularization; smaller values specify stronger regularization.
- `penalty`: Set to `'l2'` to prevent overfitting and manage high-dimensional feature spaces.
- `solver`: `'liblinear'` or `'saga'`, chosen depending on dataset size and sparsity.
- `max_iter`: Maximum number of iterations for the optimization algorithm to converge.

Logistic Regression serves as a strong baseline due to its balance of simplicity and predictive power. We will be using scikit-learn's built-in model to perform our classification implementation.

2.3.2 Model 2: Random Forest

Random Forest is an ensemble method that builds multiple decision trees during training and outputs the mode of their predictions. Each tree is trained on a bootstrapped sample of the data and considers a random subset of features at each split, which introduces diversity and reduces overfitting.

Formally, the prediction of a Random Forest classifier is:

$$\hat{y} = \text{mode}\,\{T_1(\mathbf{x}), T_2(\mathbf{x}), \ldots, T_B(\mathbf{x})\}$$

where T_i denotes the i^{th} decision tree in the forest of B trees.

Random Forests are advantageous for this problem because:

- They can model complex, nonlinear relationships between features and labels.
- They are resilient to overfitting, especially when many trees are used.
- They provide feature importance metrics, which aid interpretability.

Hyperparameters used:

- `n_estimators`: Number of trees in the forest; higher values generally improve performance up to a point.
- `max_depth`: Limits the depth of each tree to prevent overfitting.

- `min_samples_split`: Minimum number of samples required to split an internal node.
- `max_features`: Number of features considered for splitting at each node; typically set to `'sqrt'` or `'log2'` for classification tasks.
- `bootstrap`: Whether to use bootstrapping when sampling for training each tree.

Random Forests tend to outperform simpler models like Logistic Regression when the feature space exhibits nonlinear interactions, as is likely the case with complex linguistic features and statistical text properties that have been vectorized to its numerical counterparts.

2.3.3 Model 3: BERT

For our third model, we used BERT (Bidirectional Encoder Representations from Transformers), a popular language model developed by Google. BERT is known for its ability to understand language context better than older models because it reads text in both directions at once. This allows it to pick up on subtle patterns and meaning in writing.

We used the pre-trained `bert-base-uncased` model from the Hugging Face Transformers library. It has 12 layers and was originally trained on a large amount of general English text. We fine-tuned this model by adding a simple classification layer on top to predict whether an input text was written by a human or by AI.

Preprocessing. Before feeding text into BERT, we used its tokenizer to split words into smaller parts (called subword tokens) and add special markers like [CLS] and [SEP]. We also padded or cut the sequences to a consistent length and added attention masks to tell the model which parts of the input were real words and which were just padding.

Training. The model was trained with a basic setup using cross-entropy loss and the AdamW optimizer. Because BERT is a very large model, training it was slower and more demanding compared to the other models we tried. We only ran a small number of training steps due to limited resources.

Challenges. BERT was the hardest model for us to understand. It uses complex ideas like attention mechanisms and transformer blocks, which were new to us. We only implemented a basic version of BERT, mostly using existing libraries and tutorials. Even though we didn't fully understand every part of how it works, it was still helpful to experiment with such a powerful model.

2.4 Training and Evaluation Methodology

To ensure that our models generalized well to unseen data and were not overfitting, we followed a training and evaluation process.

1. Pre-processing the data and performing EDA with appropriate visualizations.
2. Vectorization of data (strings and text) into numerical vectors for classification.
3. Splitting the data into training and testing sets with an 80/20 ratio.
4. Training the baseline models on the training sets.
5. Fine-tuning our models with hyperparameter training on a grid cross-validiation search.
6. Evaluate performance with our performance metrics from scikit-learn and SHAP.

Data Splitting and Preprocessing, Vectorization

The labeled text dataset was split into training and testing sets using an 80/20 ratio via `train_test_split` from `scikit-learn`. The training data was used for both model fitting and hyperparameter tuning, while the test set was reserved for final evaluation. Text was vectorized using TF-IDF representations via `TF_IDF_Vectorizer` from `scikit-learn`, and additional engineered features (such as readability metrics and token statistics) were concatenated to create a final feature matrix.

Cross-Validation

We employed 5-fold cross-validation on the training set for hyperparameter tuning. This involved splitting the training data into five subsets, training on four of them, and validating on the fifth—rotating through all combinations. This process was repeated for each hyperparameter configuration using `GridSearchCV`, allowing us to select the best-performing model parameters in a data-efficient and robust way.

Evaluation Metrics

To assess model performance, we used the following metrics via `classification_report`:

- **Accuracy**: The proportion of correctly predicted instances of all predictions. This serves as a basic performance benchmark.
- **Precision and Recall**: Especially important in imbalanced classification tasks. Precision quantifies how many predicted AI-generated texts were correct, while recall measures how many actual AI-generated texts were identified.
- **F1 Score**: The harmonic mean of precision and recall, providing a single metric that balances the two.
- **AUC-ROC**: Evaluates model performance across all classification thresholds by plotting the true positive rate against the false positive rate. A higher AUC indicates better model discrimination capability.
- **Feature Importance**: Helps identify which of the features tested on, whether it be an engineered feature or a word vector, was given the most weight during training.

These metrics together gave us a comprehensive understanding of model performance, especially in contexts where a misclassification could have asymmetric consequences (e.g. falsely flagging a human-written essay as AI-generated).

Model Comparison

After training, we evaluated each model in the reserved test set using the metrics mentioned above. In addition to numerical comparisons, we visualized confusion matrices and ROC curves to understand the behavior of each model. We also applied SHAP (SHapley Additive Explanations) to interpret feature importance and assess whether models were making decisions based on meaningful textual characteristics.

By comparing both traditional and ensemble models under consistent conditions, we were able to identify the strengths and weaknesses of each and make informed decisions about which approaches performed best for the task of AI-text detection.

3 Results

We evaluated three models—Logistic Regression, Random Forest, and BERT—on the same classification task: determining whether a piece of text was written by a human (label 0) or generated by AI (label 1). The models were assessed using accuracy, precision, recall, and F1-score.

Logistic Regression performed very well, achieving an accuracy of 94%. It showed strong precision and recall for both classes, with a weighted F1-score of 0.94. This model benefited from a combination of TF-IDF features and engineered features such as word count and readability. Its performance, simplicity, and interpretability made it a solid choice for the task.

Random Forest delivered the highest overall performance, with an accuracy of 99%. It achieved near-perfect precision and recall (0.97–1.00), indicating extremely effective classification. This model's ability to capture non-linear relationships between text structure and class labels helped it take full advantage of the engineered features. Its high accuracy suggests that the features we created were particularly well-suited for tree-based methods.

BERT, while a state-of-the-art model, underperformed relative to the simpler models. It achieved an accuracy of 80%, with a strong recall of 0.89 for AI-generated text but a lower precision of 0.70. This led to more false positives when predicting AI content. Additionally, BERT was the most difficult model for us to work with. Due to limited time and resources, we only implemented a basic version without deeper fine-tuning, which likely affected its performance.

Model Comparison. The table below summarizes the test set performance of all models:

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
Logistic Regression	0.94	0.94	0.94	0.94
Random Forest	0.99	0.99	0.98	0.99
BERT	0.80	0.83	0.82	0.81

In summary, Random Forest was the top-performing model, with Logistic Regression close behind. BERT, despite its complexity, did not outperform the simpler models in our project. With more experience and compute power, we believe BERT could perform better in future work.

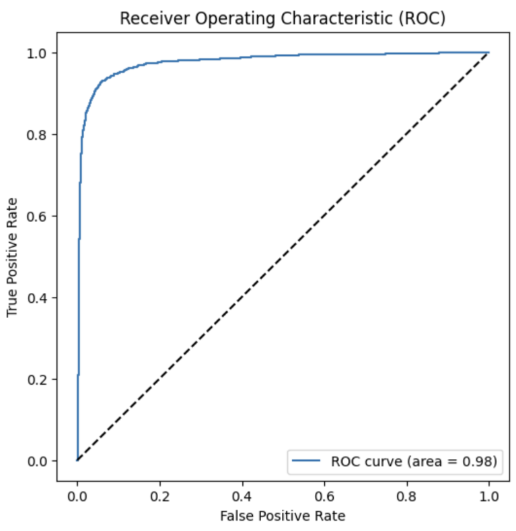


Figure 3: Logistic Regression ROC

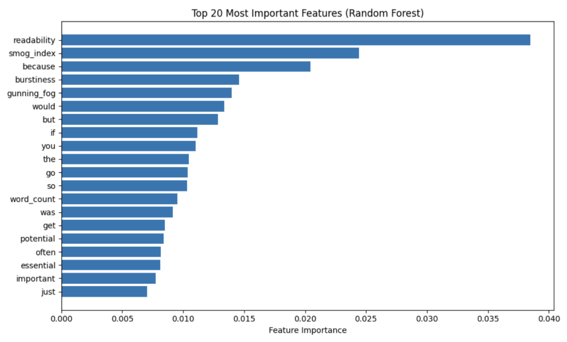


Figure 4: Random Forest Feature Importance

3.1 Comparative Analysis

In this section, we compare the three models—Logistic Regression, Random Forest, and BERT—based on their evaluation metrics, implementation complexity, and overall performance in classifying AI-generated versus human-written text.

Logistic Regression demonstrated strong and consistent performance, achieving 94% accuracy with balanced precision and recall across both classes. It was easy to implement, fast to train, and interpretable. The model worked particularly well with TF-IDF features and benefited from the engineered metrics like readability and word count. Its main strength lies in its simplicity and effectiveness, especially when paired with well-structured input features.

Random Forest outperformed all other models, reaching 99% accuracy with near-perfect classification metrics. It was highly effective at capturing non-linear relationships in the data, especially between the engineered numerical features. Random Forest's ability to rank feature importance also provided useful insights. However, it is less interpretable than Logistic Regression and can overfit if not carefully tuned—though overfitting did not appear to be an issue in our case.

BERT, while a powerful deep learning model, underperformed compared to the simpler models. It reached only 80% accuracy and showed a noticeable class imbalance in precision and recall. Its strength is in understanding deep contextual language patterns, but our limited experience with fine-tuning, combined with resource constraints, likely restricted its potential. BERT was also the most complex and computationally expensive model to train.

Best-Performing Model. Based on our results, **Random Forest** was the best-performing model overall. It combined high accuracy with strong recall and precision for both classes. Its success is likely due to how well it leveraged the engineered features and its robustness in handling structured data. While BERT may outperform in other NLP tasks with more tuning, Random Forest provided a practical and highly effective solution for this specific classification problem.

Overall, this comparative analysis shows that classical machine learning models—when paired with thoughtful feature engineering—can rival or even outperform more complex deep learning models under the right conditions.

4 Discussion

In the context of our research quetion, our results have demonstrated that traditional machine learning models—particularly Logistic Regression and Random Forest—can achieve competitive performance in classifying AI-generated versus human-written text. This suggests that with careful feature engineering, simpler models can offer strong baseline performance on this problem, even without the computational complexity of deep learning models. However, there are multiple underlying factors that can influence this, such as the nuances of our data being too "simple" or too small. This allows our models to shine, outperforming our baseline-BERT model, which was the goal of our project.

The engineered features, such as readability scores, burstiness, and sentence structure complexity, played a crucial role in enhancing the models' discriminatory power. These features seem to capture stylistic and structural differences between human and AI-generated text that are not always obvious through raw token frequency alone.

4.1 Connections to Prior Work

Prior work in this domain, especially models based on large language transformers like BERT and GPT detectors, has emphasized the power of deep contextual embeddings. While our baseline BERT implementation provides a strong performance reference, our project shows that models like Logistic Regression can approach (and in some cases match or outperform) such baselines when paired with well-curated features.

Additionally, many existing tools for AI-text detection are opaque or proprietary. Some of the tools we took inspirationd from:

- GPT-Zero
- SynthID
- Content Credentials
- DetectGPT

Our work contributes to the growing body of open, interpretable approaches that offer transparency in how decisions are made.

4.2 Limitations

There were quite a few limitations that had inhibited our project from being a fully-deployable AI-detector tool.

Limitation	**Impact / Explanation**
Dataset Scope	Our dataset may not generalize well to emerging models or real-world deployment environments, especially when handling adversarial text. Training on larger datasets could help remedy this, given more time.
Feature Ceiling	Shallow features may overlook deeper linguistic or contextual nuances, so we could implement more engineered features to better capture semantics eg. [Perplexity, Punctuation Variety, Word Stemming, Type-Token Ratio].
Robustness	We did not have the time to enhance our models, evaluating against adversarial or paraphrased inputs requires self-augmented data.
BERT Model	The baseline BERT model was outperformed, most likely due to a lack of fine-tuning the parameters–which requires a better understanding of large language transformers.

Table 1: Summary of Key Limitations

5 Conclusion & Future Work

In this project, we built and evaluated machine learning classifiers to distinguish between AI-generated and human-written text. Using a combination of TF-IDF features and engineered linguistic statistics, we trained Logistic Regression and Random Forest models that performed competitively with baseline transformer models like BERT.

Our findings highlight that interpretable, lower-resource models can still be viable for AI-text detection, especially when enhanced with meaningful feature engineering. This opens up possibilities for lightweight, explainable tools that can be deployed in settings that lack computational power and need faster, more efficient, interpretable detector models.

Future Implementation

Looking ahead at features of our project we still hope to implement in the future given the time and resources:

- **Scale training data:** Train on a larger and more diverse dataset of over 1.39 million rows, including augmented and self-generated examples.
- **Enhance feature engineering:** Introduce additional linguistic and semantic features that go beyond surface-level statistics.
- **Refine BERT-based models:** Perform deeper fine-tuning of transformer-based models (e.g., BERT) with improved pre-processing and task-specific adaptations to increase classification accuracy.
- **Build ensemble classifiers:** Experiment with ensembling techniques that combine diverse model types—such as Multi-Layer Perceptrons (MLPs), Gradient Boosting Machines, and Random Forests—to leverage their complementary strengths and build an aggregated accuracy score and predictor.
- **Improve robustness to evasion:** Evaluate and adapt models to withstand adversarial inputs, such as paraphrased or deliberately obfuscated AI text, which we could train using self-augmented/self-generated datasets.
- **Deploy interactive tools:** Create a lightweight dashboard or web application that allows users to to view visualizations regarding performance evaluation, and to feed the application text to receive a prediction of whether or not the text is AI-generated.

Acknowledgments

We would like to thank Professor Kelsey Urgo for her help and constructive feedback on our idea along the way, giving us ideas to further improve our project-line. She told us about using BERT, to which we then came up with the problem statement of beating a baseline-BERT implementation when comparing performance metrics.

References

Kaggle. Llm - detect ai generated text dataset, 2023. URL `https://www.kaggle.com/datasets/sunilthite/llm-detect-ai-generated-text-dataset`. Last accessed 5/06/2025.

Kaggle [2023]

A Appendix: Additional Visualizations (Optional)

B Appendix: Code Implementation Details (Optional)

B.1 Implementing User Input

Our code snippet and result for being able to input any user-generated text string, and have our model predict its class (AI or Human):

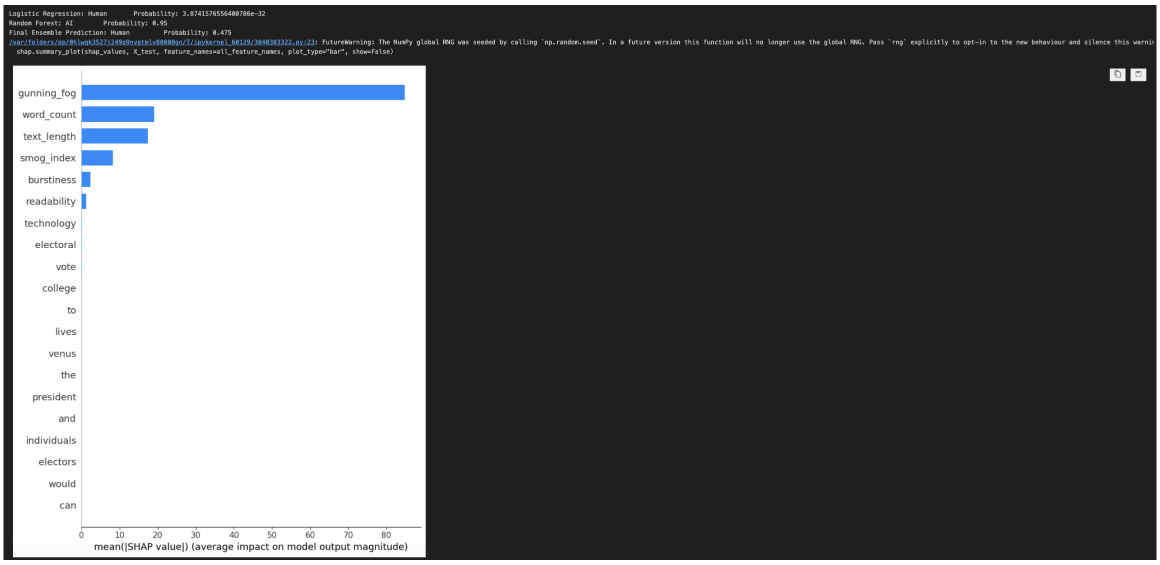
```
## definining the functions to predict user input as an ensemble of models

## vectorizing input text
def preprocess_input(text, vectorizer):
    X_tfidf = vectorizer.transform([text])
    X_engineered = extract_engineered_features(text)
    return hstack([X_tfidf, X_engineered])

## prediting with our models, generating confidence scores
def predict_ensemble(text, vectorizer, model_lr, model_rf):
    X_input = preprocess_input(text, vectorizer)

    pred_lr = model_lr.predict(X_input)[0]
    prob_lr = model_lr.predict_proba(X_input)[0][1]

    pred_rf = model_rf.predict(X_input)[0]
    prob_rf = model_rf.predict_proba(X_input)[0][1]

    votes = [pred_lr, pred_rf]
    final_pred = int(np.round(np.mean(votes)))
    final_conf = np.mean([prob_lr, prob_rf])

    return {
        "Logistic Regression": (pred_lr, prob_lr),
        "Random Forest": (pred_rf, prob_rf),
        "Final Ensemble Prediction": (final_pred, final_conf)
    }


def extract_engineered_features (text):
    burstiness = calc_burstiness(text)
    readability = textstat.flesch_reading_ease(text)
    word_count = len(word_tokenize(text))
    text_length = len(text)
    gunning_fog = np.clip(textstat.gunning_fog(text), 5, 20)
    smog_index = textstat.smog_index(text)
    return np.array([[burstiness, readability, word_count, text_length, gunning_fog,
    smog_index]])
```

Listing 1: Predictive RF & LR Models Taking User Input to Classify Text

```
## INPUT TEXT HERE
sample_text =  "The rapid advancement of technology continues to reshape industries
    and influence the daily lives of individuals across the globe."

## model
from scipy.sparse import csr_matrix
result = predict_ensemble(sample_text, vectorizer, model, rf_clf)
label_map = {0: "Human", 1: "AI"}
for model_name, pred in result.items():
    print(f"{model_name}: {label_map[pred[0]]}\t Probability: {pred[1]}")


'''explainer = shap.Explainer(model.predict, X_train)
shap_values = explainer(csr_matrix(preprocess_input(sample_text, vectorizer)))
shap.plots.waterfall(shap_values[0])'''

explainer = shap.LinearExplainer(model, X_train)
shap_values =  explainer.shap_values(csr_matrix(preprocess_input(sample_text,
    vectorizer)))

tf_idf_features = vectorizer.get_feature_names_out().tolist()
engineered_features = ['text_length', 'word_count', 'readability', 'gunning_fog', '
    smog_index', 'burstiness']
all_feature_names = tf_idf_features + engineered_features

shap.summary_plot(shap_values, X_test, feature_names=all_feature_names, plot_type="bar
    ", show=False)
plt.tight_layout()
plt.show()
```

Listing 2: User Input

Associated Output:



```
Logistic Regression: Human       Probability: 3.8741576556400786e-32
Random Forest: AI        Probability: 0.95
Final Ensemble Prediction: Human         Probability: 0.475
/var/folders/pp/0hlwqk3527j249q9nvgtmlv80000gn/T/ipykernel_60129/3040383322.py:23: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future version this function will no longer use the global RNG. Pass `rng` explicitly to opt-in to the new behaviour and silence this warnin
  shap.summary_plot(shap_values, X_test, feature_names=all_feature_names, plot_type="bar", show=False)
```

Figure 5: Interactive Result User Input