Machine Learning Models to Differentiate Humans from AI

Gregory Standish-Butcher
gas9901@nyu.edu
Nawab Mahmood
nm3749@nyu.edu
Suhan Suresh
ss15182@nyu.edu

Abstract

The primary goal of this research is to develop a model capable of differentiating text produced by ChatGPT 3 or 3.5 from human generated text. To do this we feature extract and preprocess data. This uses Spacy for NLP analysis and Numpy for calculations on metacontext. We then create a BERT-based custom neural network model. This model uses Dask disk partitioned data frames, Early Stopping, AdamW, AutoCast, and GradScalar. The Dataset we use is AI Vs Human Text by Shayan Gerami. Evaluation results demonstrate significant improvements in accuracy and scalability compared to existing methods, paving the way for future research and analysis."

Introduction

The growing prevalence of AI-generated text poses a unique challenge in distinguishing it from human-authored content. In the misinformation and media age, it has become a complex challenge in our everyday lives. Our research focuses on detecting AI-generated text to counteract misinformation and non-authentic content. This generation of generative content comes with extensive risk and has the possibility to be extremely dangerous in the future. Models are also increasing in capability at a very rapid pace. As a result, research that focused on detecting earlier AI-generated content is no longer able to detect ChatGPT 3. GPT and other AI models are improving and building off themselves. We must do the same for their detection, building off previous research and improving. Our goal is therefore to create a model capable of differentiating text produced by ChatGPT 3 and 3.5.

Previous Works and Research

1. Christine P. Chai in her research paper *Comparison of Text Preprocessing Methods* provides a thorough analysis of the challenges associated with preprocessing. Chai's research shows how the unique properties of datasets like language, domain, and structure can significantly influence the required preprocessing techniques. She shows us that a one-size-fits-all approach to processing text is not the best approach in the context of AI detection; the methods used should depend on the exact research, data, and language.
Throughout our process we faced many similar challenges to what Chai describes in her research. One major difficulty being the cleaning and preprocessing of the dataset. Datasets

containing technical or complex language often require specialized tokenization and text normalization methods to be properly understood. Informal data on the other hand, such as social media, presents further challenges. The use of informal language, slang, and non-standard spelling or punctuation practices cause formatting issues and misspellings. These formatting issues are also different for every data set being worked with, making it hard to generalize our cleaning and preprocessing.

2. The article *Comparing BERT against traditional machine learning text classification* by Santiago González-Carvajal and Eduardo C. Garrido-Merchán provides insights into the effectiveness of Google's BERT model in text classification tasks. The authors conducted four experiments to compare the performance of BERT against traditional machine learning approaches that utilize TF-IDF features. These experiments covered a diverse range of text classification tasks, including sentiment analysis on movie reviews (IMDB dataset), tweet classification (RealOrNot dataset), news article classification in Portuguese, and sentiment analysis on Chinese hotel reviews.

Model	Accuracy
BERT	**0.9387**
Voting Classifier	0.9007
Logistic Regression	0.8949
Linear SVC	0.8989
Multinomial NB	0.8771
Ridge Classifier	0.8990
Passive Aggresive Classifier	0.8931

Table 1. Accuracy retrieved by the different methodologies in the IMDB experiment over the validation set.

BERT repeatedly achieved a higher accuracy in a shorter training time. It also did not require extensive feature engineering. This suggests that BERT is capable of capturing the nuances and patterns in text data, making it a promising approach for distinguishing AI-generated text from human-generated text.

The article also highlights the importance of transfer learning and pre-training in BERT's success. By leveraging pre-trained weights and fine-tuning on specific tasks, BERT can achieve high accuracy even with relatively small datasets. This is crucial for our research, as it demonstrates the potential of using pre-trained BERT models and fine-tuning them for the task of identifying AI-generated text. This article demonstrated BERT's superior performance, justifying it as the primary choice of model for our research.

3. The research paper *TensorFlow: A System for Large-Scale Machine Learning* by Martín Abadi gives insights into the design and implementation of Google's TensorFlow, a powerful and

flexible system for large-scale machine learning. The paper discusses the challenges involved in designing a system that can handle the unique characteristics of machine learning workloads, particularly in the context of deep learning. TensorFlow uses a unified dataflow graph to represent the computation and state in a machine learning algorithm. This enables the system to handle a wide range of models and optimize their execution across other hardware. This allows for hardware acceleration and massively aids computational efficiency. We found this to be vital in our research.

Tensorflow also incorporates fault tolerance mechanisms. Training deep learning models on large datasets can be time-consuming and resource-intensive. Any interruptions can lead to significant setbacks. By utilizing TensorFlow's checkpointing and recovery features, we can protect ourselves from data loss and minimize the impact of system failures on our research progress. This will help us maintain the integrity of our results, which is crucial for progress.

Datasets Used

Our initial dataset selection was based on the HC3 dataset, well-known for its extensive use in evaluating text generation models. However, we encountered significant challenges that impacted the dataset's practicality for training a model to differentiate between human-generated and AI-generated texts effectively. A major issue we faced was the inconsistency in formatting between the human-generated and AI-generated (GPT) texts within the HC3 dataset. Specifically, the AI texts contained unique formatting characters like newline characters (‘\n’) that were absent from the human texts. This discrepancy introduced a significant bias as the model began learning to identify text sources based on these superficial formatting cues rather than the actual linguistic content and properties of the text itself.

Example extracts from HC3:
“chatgpt_answers” :[“ \nBanks offer a number of financial services …”]
“human_answers” :[“ \/ information \/ details etc . do n’t get cut …”]
The HC3 dataset is predominantly composed of data from specific internet forums like Reddit's ELI5 (Explain Like I'm Five), where an informal style with slang is prevalent. This lack of variety posed a problem, as the model tended to overfit to this informal style rather than generalizing across a broader linguistic spectrum. Therefore, unless it was explicitly trained on informal language, the model could easily misclassify well-written formal GPT text as human-generated. Due to these challenges, we sought an alternative dataset, ultimately selecting AI Vs Human Text by Shayan Gerami.

Data Collection and Preprocessing

Our methodology begins with data preprocessing. Here we turn basic CSV and JSON files into lists of feature extracted and labeled JSON packets to be fed into our neural network model. We separate the AI and Human data so that we can load in an even split of AI and Human data during training. The basic overview of our preprocessing is:

1. Undersampling
2. Cleaning
3. Parts of Speech (POS) Feature Extraction
4. Further Feature Extraction
5. Packaging the finished entry into a new JSON file

Step 1. Undersampling

Ideally, our dataset should not be biased in any way. We therefore undersample any new dataset before development to contain equal amounts of each class time. If we used heavily one sided data, the Neural Network could easily become biased and produce some false initial positive results by simply giving a higher probability to the more prominent class type. This of course would not generalize and might actually make it harder for the model to train as it would have to get past this local minima to further increase in accuracy.

Step 2. Cleaning

As written in our Data section, we go through an extensive cleaning process. We use regex statements to remove:

- Double spaces
- Unnecessary spaces or slashes around punctuation
- Any escape sequences such as '\n'
- Any non alphanumeric non punctuation characters e.g. â, ë, œ

We then use *pyspellchecker* to check for misspelled or indiscernible words in the text. If too many indiscernible words are found it removes that entry from the dataset. This prevented the model from leveraging these superficial cues as predictive features rather than using the text's actual linguistic properties and content.

Step 3. Parts of Speech (POS) Feature Extraction

We tried various NLP feature extraction techniques with successful improvements to learning from many different cases. The main bottleneck we faced during this stage of development was the computational cost of preprocessing and model training required to evaluate each technique. We used smaller datasets but that often lost precision in our evaluation. We also faced a slower

rate of learning when trying to encode many different feature types - although this may be solved with longer training times in future. In response, we decided to focus on adjectives and adverbs. We used the python library *spacy* to tokenize and POS tag the text. We then copied the words tagged as ‘ADJ’ or ‘ADV’ to a new features dictionary. We believe extracting adverbs and adjectives can provide insights into the descriptive style of text passages, and therefore detect a particular common style of ChatGPT generated txt.

Step 4. Further Feature Extraction

We also tried to include other feature extraction techniques. We found that sentence length analysis performed well. It also had a low computational overhead. Here we took the average length of sentences in the text extract and used it to calculate the standard deviation of sentence length within the text. We then added both of these figures to the features dictionary.

Step 5. Packaging the finished entry into a new JSON file

The features dictionary is then saved as part of the json packet such that the final product is:


```
{
   "text": "In Winston Churchill…",
   "features": "{"adverbs": "subsequently …",
                "adjectives": "crucial …",
                "avg_sentence_length": 27.0,
                "sentence_length_variability": 12.649110640673518}",
   "label": 1
}
```


These preprocessing steps are a crucial step for enhancing the model's ability to distinguish meaningful patterns and features within the textual data. They increase the efficiency of learning and help the model to reach higher accuracies without overfitting.

The Model Configuration

Model Customization

We start with the famous Bidirectional Encoder Representations from Transformers (BERT) model, commonly used for text and nlp based applications. To properly accommodate our new feature input we must customize the model to take two text inputs and two numerical inputs. You can see this in Figure 1 below. The original bert model is used to process the text based features, and the final layer processes the output of bert as well as the numerical features. This helps keep the computation done on the numerical features low, reducing the chance of overfitting.

```python
class CustomBertModel(nn.Module):
    def __init__(self, bert_model_name, numerical_feature_size, device):
        super(CustomBertModel, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name).to(device)
        self.fc = nn.Linear(self.bert.config.hidden_size + numerical_feature_size, 1).to(device)

    def forward(self, input_ids, attention_mask, numerical_features):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = outputs[1]
        # Concatenate the pooled output with numerical features
        combined_output = torch.cat((pooled_output, numerical_features), dim=1)
        logits = self.fc(combined_output)
        return logits
```

Figure 1: Code Extract - Custom Neural Network Model

Tokenization and Encoding

The text based data is first tokenized by BERT's inbuilt tokenizer. The text and features are independently tokenized and encoded. Ideally this emphasizes the extracted adverbs and adjectives making them more useful. BERT takes a maximum 512 tokens as its input which means each encoding must share this space when either being truncated or padded by the tokenizer. We decided the text would be allowed 400 tokens and the adverbs and adjectives 112. Unfortunately this means any extra text is discarded, however by allowing the extracted feature space more tokens, the adverbs and adjectives of the extra text can be salvaged. Conversely if the text of adverb adjective space is ever less than the max length it is padded. This ensures that the model always sees the adjectives and adverbs section in the same section of its input.

Hyperparameters and Optimization

We used Binary Cross Entropy for loss, which calculates the difference between the predicted probabilities and the actual class labels in a dataset using the following equation:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

Equation 1: Binary Cross Entropy Loss Equation

We used AdamW for our optimizer. AdamW is different from the standard Adam (Adaptive Moment Estimation) optimizer because it decouples weight decay from the optimization process. This means the learning rate and weight decay are optimized separately, helping AdamW's precision when tuning. This is also supported by pytorch's gradient scalar, which optimizes training by dynamically adjusting the scale of gradient calculations to maintain precision and maximize computational efficiency.

Model Training

The data is loaded in from the preprocessed json packets. As preprocessing separates the AI and Human classes, we can undersample again to ensure that the same number of AI and human data entries have been loaded. The data is then combined and shuffled.

Validation Data is randomly selected from the dataset and removed. The remaining data is then split into chunks for each epoch so that no data is reused or remembered. This is to prevent overfitting. After every epoch the model is evaluated on the validation dataset (however it does not remember this data or learn so the data can be used repeatedly).

This evaluation is used to power early stopping. If the model does not see an increase to its validation loss for 3 epochs, it will save the best model and quit. This speeds up development time by a massive degree, reducing training time by anywhere from 50% to 80%. The validation evaluation for our currently used model produced these results:

	precision	recall	f1-score	support
Human	0.96	0.93	0.95	1004
AI	0.92	0.96	0.94	863
accuracy			0.94	1867
macro avg	0.94	0.94	0.94	1867
weighted avg	0.94	0.94	0.94	1867

Figure 2: Validation Set Results

Results

Results were measured on an untouched subset of the preprocessed data called the testing set. To ensure validity we split the preprocessed data into a training set and a testing set. The model was solely exposed to the training set during development and only given the testing set at the end to produce the graphics below. This set is different to the validation set, which is produced from the training set for every new model creation.

We loaded 2500 testing set entries from each class for a total of 5000. We used a macro F-score as we have no bias towards one class or the other. Below are the results:

	precision	recall	f1-score	support
Human	0.96	0.93	0.94	2500
AI	0.93	0.96	0.95	2500
accuracy			0.94	5000
macro avg	0.95	0.94	0.94	5000
weighted avg	0.95	0.94	0.94	5000

Figure 3: Evaluation Set Results

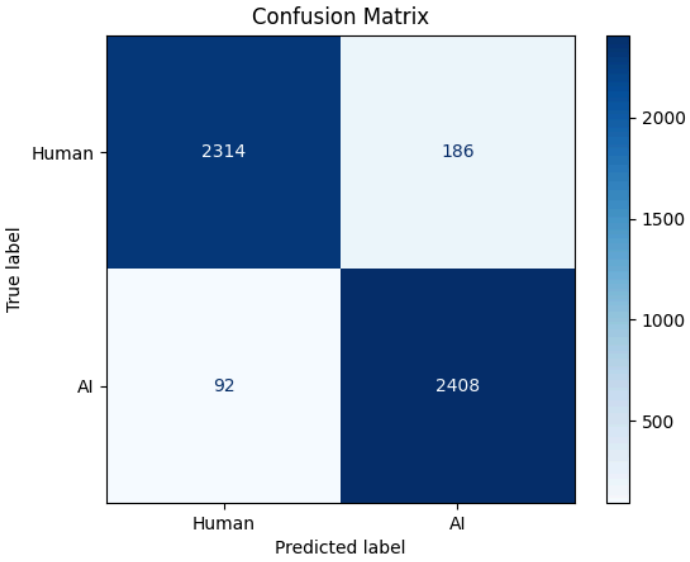

Figure 4: Prediction Success Confusion Matrix

.

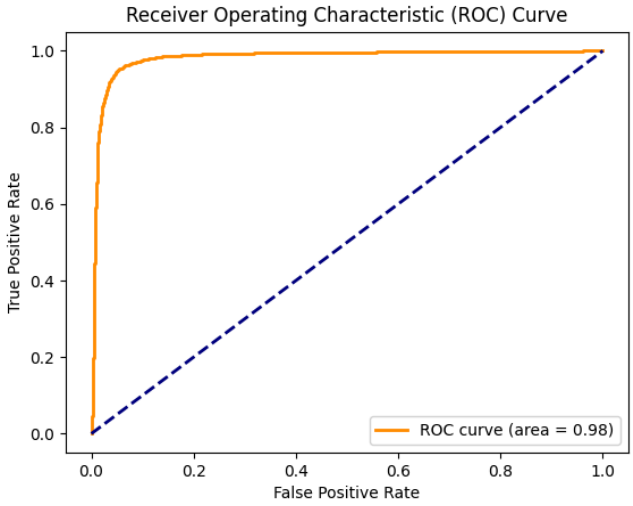

Figure 5: ROC curve Graph

These results are satisfactory and meet the project's accuracy goals. The precision, recall, and f-scores remained almost the same as the validation set, clearly showing we had successfully mitigated the high risk of overfitting. With an F-scores of 0.94 and 0.95 and an ROC curve of 0.98, we believe the overall performance of our model is very high.

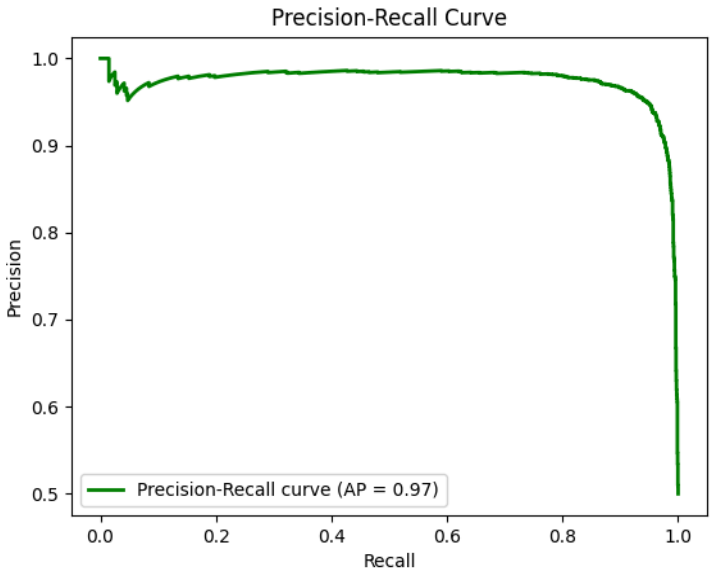

Figure 6: Precision over Recall Graph

Looking at the precision-recall trade off, we see a small dip in precision at the low recall mark - we are actually unsure of the cause of this - but an otherwise flat graph showing little variation. The average precision across all recall points is 0.97.

Comparable Analysis

Niful Islam in his research titled *Distinguishing Human Generated Text From ChatGPT Generated Text Using Machine Learning* used a dataset consisting of 10,000 texts. 5,204 texts were human generated, collected from news and social media. The remaining texts were generated by GPT-3.5. The authors encountered similar challenges with formatting inconsistencies between human-generated and AI-generated texts. They also addressed this through data preprocessing techniques. They also list using one-hot encoding which we did not implement.

Their results found interestingly that deleting stop words negatively impacted the classification performance, as the selection of stop words plays a crucial role in differentiating human and AI-generated text. Their proposed ERTC (Ensemble of Random Trees Classifier) model achieved an accuracy of 77%. The authors conducted a comparative analysis of 11 machine learning and deep learning algorithms, revealing that some well-known classifiers, such as K-Nearest Neighbor and Decision Tree, performed poorly on this task.

In comparison, we managed to achieve a higher accuracy of 94%. We did not however have the computational power to evaluate many different techniques, models, or methods. We did not try to remove stop words, but we did extract additional features as part of our preprocessing. Our dataset was also much larger at over 400,000 entries.

Our research relates well with Islam's as we both struggled with publicly available datasets. These datasets are rich in volume but contain numerous errors and biases that can significantly affect the accuracy of AI detection systems. Working with noisy data that contains inconsistent formatting, typos, and slang, challenges the model's ability to learn effective patterns for distinguishing between human-generated and AI-generated text. This is again similar to what was mentioned in Chai's research.

Possible Improvements To Our Model

Greater access to computation would greatly improve our abilities to test ideas and train our model. We had many other POS-based and meta contextual features to test and evaluate. It would also be interesting to see if early stopping did stop us from finding a greater absolute minimum, or whether further epochs would have yielded no results.

The biggest improvement we could make is to the data. It is clear that the model is capable of learning indicators within the text, especially with our preprocessing techniques. The more data available to be mixed in and combined, the greater the breadth of AI written work the model can be trained to detect. We also did not find any sufficient data on ChatGPT 4. As a result the model can not accurately detect these more advanced models.

We also would like to see data that had a mix of ChatGPT and human writing, or human writing that had been heavily edited by AI. This data would be more challenging to find and categorize but may provide a more useful model if successful.

We could also improve the model in many ways. Further feature extraction tests, providing we had greater computational power to train and evaluate many different models, could provide a better preprocessing stage. Finally, trying different customisations on the model and tuning the hyperparameters could still be explored indefinitely, provided more time and computation, this may provide a better model for generalized detection tasks.

Conclusion

We used preprocessing and Google's BERT model to build an AI text detection system. It takes in text and gives a reading from 0 to 1. We used SHAYAN GERAMI's dataset to train and test our model. It proved relatively successful, reaching an average accuracy of 94%. We believe this technology is vital in the age of fake data and misinformation. Islam's work and our research highlight the importance of developing machine learning-based approaches to detect AI-generated text. As the capabilities of large language models like ChatGPT continue to evolve, it is crucial to establish new methods for ensuring the authenticity of textual content. We can only do this by building upon existing research. Our aim is to contribute to the advancement of this field and provide practical solutions for distinguishing between human-generated and

AI-generated text. We believe this model could be developed further and built upon to create a detection system able to spot a broader range of AI generated text.

References

Chai, C. P. (2023). Comparison of text preprocessing methods.
Natural Language Engineering, 29(3), 509-553.
https://doi.org/10.1017/S1351324922000213

González-Carvajal, S., & Garrido-Merchán, E. (2020). Comparing BERT against traditional machine learning text classification.
Journal of Computational and Cognitive Engineering, 2.
https://doi.org/10.47852/bonviewJCCE3202838

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., & Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems.
ArXiv:1603.04467.
https://doi.org/10.48550/arXiv.1603.04467

Islam, N., Sutradhar, D., Noor, H., Raya, J. T., Maisha, M. T., & Farid, D. M. (2023). Distinguishing human generated text from ChatGPT generated text using machine learning (Version 1)
ArXiv:2306.01761.
https://doi.org/10.48550/arXiv.2306.01761

Gerami, S. (2024). AI vs Human Text: 500K AI and Human Generated Essays.
Kaggle.
https://www.kaggle.com/datasets/shanegerami/ai-vs-human-text/data