Pluralsight

Comment détecter le texte gé par l'IA : éthique et mise en œ

Apprenez à détecter le texte généré par l'IA, les outils et techniques act détection par IA, ainsi que les défis liés à l'utilisation responsable de la considérations éthiques.

Par Axel Sirota

15 avr. 2025 • 10 minutes de lecture

Guides IA & Données

Abonnez-vous à la newsletter →

À mesure que l'intelligence artificielle générative (IA géné dans la création de contenu, sa capacité à produire un te est à la fois excitante et déstabilisante. Les systèmes mod et cohérente qui suscite de légitimes inquiétudes quant à

savoir si—et comment—leur utilisation devrait automatiquement disqualifier les soumissions académiques ou professionnelles. Enfin, nous examinons les rapports récents évaluant l'efficacité des outils actuels, critiquons l'état de l'art et exposons comment ces outils pourraient s'améliorer au fil du temps.

Table des matières

L'essor de l'IA générative et ses défis

Les avancées récentes dans le **traitement du langage naturel (NLP)** ont permis de produire des systèmes d'IA capables de générer des textes sophistiqués. Des modèles tels que GPT-3.5 et GPT-4 d'OpenAI ont été déployés pour des applications allant de la rédaction d'emails et la narration créative à la rédaction d'essais académiques et de documents techniques. Cependant, à mesure que la frontière entre contenu généré par l'humain et celui généré par l'IA s'estompe, plusieurs risques apparaissent :

- **Désinformation et plagiat** : Le contenu généré par l'IA peut involontairement diffuser de fausses informations ou imiter de près des contenus existants.
- **Erosion of trust**: Educators and employers struggle to verify authorship, as traditional plagiarism checkers often miss AI-generated text.
- **Ethical and legal concerns:** Automated tools may introduce bias or lack transparency, especially if used as the sole basis for judgment.

These issues underscore the importance of developing robust techniques to detect AI-generated text.

Technical approaches to AI text detection

- **Consistent style**: Unlike the natural variation in human writing, AI tends to be more uniform in tone and rhythm.
- **Limited contextual depth**: Even when grammatically correct, AI text may miss the nuance, subtext, or originality found in human work.

Understanding these patterns helps shape effective detection strategies.

Detection techniques

Detection tools typically combine machine learning with statistical analysis:

- **Transformer-Based Classifiers**: These classifiers are developed by fine-tuning pre-trained models (e.g., RoBERTa) on labeled datasets to learn the stylistic differences between AI-generated and human-written texts.
- **Statistical Methods**: Metrics such as perplexity quantify how "surprised" a language model is by a given text. Lower perplexity indicates text that follows predictable patterns—often a sign of AI generation.
- **Low-Rank Adaptation (LoRA)**: LoRA is an efficient fine-tuning method that injects low-rank matrices into pre-trained models, allowing for task-specific adaptation with fewer parameters and reduced computational cost.

Implementation overview

Our approach combines two complementary components:

1. **LoRA-Finetuned RoBERTa Classifier:** Using the andythetechnerd03/AI-human-text dataset, we fine-tune a **RoBERTa** model with **LoRA**. This classifier learns the subtle linguistic cues that distinguish AI-generated text from human writing.
2. **Perplexity-Based Module:** We use a separate language model (e.g., GPT-2) to calculate the **perplexity of a text.** Since AI-generated text tends to be more predictable, it usually results in lower perplexity scores compared to human text. By establishing a threshold based on known human-written samples, we can flag texts with unusually low perplexity as likely AI-generated.

Combining these two methods improves robustness. The classifier provides a learned probability score, while perplexity offers an independent statistical measure. Even if one method is fooled (for example, via adversarial paraphrasing), the other may still detect the anomaly.

Detailed code steps

Pluralsight

```
from datasets import load_dataset, load_metric
from transformers import (AutoTokenizer,
AutoModelForSequenceClassification,
Trainer, TrainingArguments, DataCollatorWithPadding)
import torch
from peft import get_peft_model, LoraConfig, TaskType

# Step 1: Load the dataset
dataset = load_dataset("andythetechnerd03/AI-human-text")
# Assume the dataset includes columns "text" and "label"

# Step 2: Tokenize and preprocess the data
model_checkpoint = "roberta-base"
tokenizer =
AutoTokenizer.from_pretrained(model_checkpoint)

def preprocess_function(examples):
    return tokenizer(examples["text"],
truncation=True, padding="max_length", max_length=128)

encoded_dataset =
dataset.map(preprocess_function, batched=True)
encoded_dataset =
encoded_dataset.remove_columns(["text"])
encoded_dataset =
encoded_dataset.rename_column("label", "labels")
encoded_dataset.set_format("torch")

if "train" not in encoded_dataset.keys() or "validation"
not in encoded_dataset.keys():
    encoded_dataset =
encoded_dataset["train"].train_test_split(test_size=0.2)

# Step 3: Load a pre-trained model for classification
model =
AutoModelForSequenceClassification.from_pretrained
(model_checkpoint, num_labels=2)

# Step 4: Apply LoRA for efficient fine-tuning
lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    inference_mode=False,
    r=8,
    lora_alpha=32,
    lora_dropout=0.1
)
model = get_peft_model(model, lora_config)

# Step 5: Set up training arguments and metrics
training_args = TrainingArguments(
    output_dir="./lora-finetuned-model",
```

Pluralsight

```
    logging_steps=50,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
)
metric = load_metric("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions,
references=labels)

data_collator = DataCollatorWithPadding(tokenizer)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=encoded_dataset["train"],
    eval_dataset=encoded_dataset["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

# Step 6: Train and evaluate the model
trainer.train()
results = trainer.evaluate()
print("Evaluation results:", results)

# Step 7: Save the model
model.save_pretrained("./lora-finetuned-model")
tokenizer.save_pretrained("./lora-finetuned-model")


# ------------------------------------------------------------
# Perplexity Module using GPT-2

import torch
from transformers import GPT2LMHeadModel,
GPT2TokenizerFast

# Load GPT-2 for perplexity calculation
gpt2_model =
GPT2LMHeadModel.from_pretrained("gpt2")
gpt2_tokenizer =
GPT2TokenizerFast.from_pretrained("gpt2")
gpt2_model.eval()  # Set model to evaluation mode

def calculate_perplexity(text):
    # Tokenize the text
    inputs = gpt2_tokenizer(text, return_tensors="pt")
```

Pluralsight

```
    return perplexity.item()

# Example: Calculate perplexity for a sample text
sample_text =
"This is a sample sentence to compute perplexity."
print("Perplexity:", calculate_perplexity(sample_text))
```

Explanation of code steps

Step 1: Loading the dataset

- **Action**: We load the dataset from Hugging Face using `load_dataset("andythetechnerd03/AI-human-text")`.
- **Purpose**: This dataset contains labeled examples of AI-generated versus human-written text, which is critical for supervised training.

Step 2: Tokenization and preprocessing

- **Action**: We load the `roberta-base` tokenizer and define a function to tokenize texts with truncation and padding.
- **Purpose**: Tokenization converts raw text into numerical tokens that the model can process. Consistent input length ensures smooth batch processing.

Step 3: Loading a pre-trained model for classification

- **Action**: We load a pre-trained `roberta-base` model with a classification head for two labels.
- **Purpose**: Leveraging a model pre-trained on vast amounts of text allows us to fine-tune it on our specific classification task with higher efficiency.

Step 4: Applying LoRA for efficient fine-tuning

- **Action**: We configure and apply **LoRA** to the model using `get_peft_model`.
- **Purpose**: **LoRA** adapts only a small subset of parameters (via low-rank matrices), reducing computational cost while maintaining performance.

Step 5: Setting up training arguments and metrics

- **Action**: We define training parameters (learning rate, batch size, epochs, etc.) and specify an accuracy metric.

- **Purpose**: This process adjusts model weights to accurately classify texts as AI-generated or human-written, with evaluation ensuring the model's reliability.

Step 7: Saving the model

- **Action**: The fine-tuned model and tokenizer are saved for future use.
- **Purpose**: Saving the trained model allows for deployment without re-training from scratch.

Perplexity Module using GPT-2

- **Action**: Load a GPT-2 model and its tokenizer.
- **Action**: Define a function `calculate_perplexity` to compute the perplexity of a given text.
- **Purpose**: Perplexity measures how predictable the text is. Lower perplexity suggests text that aligns with patterns typical of AI-generated content.
- **Purpose**: This independent statistical metric complements the classifier's output, strengthening overall detection reliability.

Ethical considerations, limitations, and responsible implementation

Despite promising technological advances, significant ethical challenges remain:

- **False positives and negatives**: No detection tool is perfect. Incorrectly flagging genuine human writing or missing cleverly modified AI text can lead to severe consequences, such as wrongful academic or professional sanctions.
- **Bias against marginalized groups**: Some detection systems have been found to disproportionately flag work by non-native English speakers or neurodivergent individuals, potentially exacerbating existing inequities.
- **Lack of transparency:** Many tools operate as "black boxes" with little explanation for their decisions, making it hard for those affected to understand or contest them.

To mitigate ethical risks, detection systems should:

- **Use AI tools as support, not final judgment**: Automated results should prompt further human review rather than serve as the sole basis for punitive decisions.
- **Adopt a holistic evaluation approach**: Combine detection outputs with traditional assessment methods (e.g., in-class writing, oral examinations, revision history analysis) to verify authenticity.

As generative AI continues to evolve, detection tools must keep pace—pushing toward smarter, fairer, and more adaptable approaches.

- **Enhanced detection algorithms** - The field is evolving, with several promising developments on the horizon:
  - Multi-Modal and Ensemble Methods: Future detectors may integrate textual analysis with additional data (for example, keystroke dynamics or revision history) and combine multiple detection strategies for a more comprehensive approach.
  - Resilient Watermarking: Advanced watermarking techniques aim to embed robust, imperceptible markers in AI-generated text that remain detectable even after adversarial modifications.
  - Adaptive Learning Systems: Continuous learning frameworks will allow detection models to update in real time as adversaries develop new evasion techniques.
- **Integration with educational and professional platforms -** Detection tools will likely become more integrated with existing systems:
  - Learning Management Systems (LMS): Tools integrated into platforms like Google Docs or institutional LMS can track revision histories and verify the authenticity of the writing process.
  - Holistic Assessment Tools: By combining automated analysis with manual evaluation and other performance metrics, institutions can create a balanced approach that supports both fairness and academic integrity.
- **Transparency and explainable AI:**
  - Explainable Models: Future systems should provide not only detection outcomes but also clear explanations for why a text was flagged, thereby enhancing trust and accountability.
  - Open-Source Initiatives: Open access to model code and datasets for independent audits can foster community-driven improvements and help identify biases.

Conclusion and Outlook

Detecting AI-generated text is a complex, multifaceted challenge that demands a careful balance between advanced technological methods and strong ethical oversight. By combining a LoRA-finetuned RoBERTa classifier with a perplexity-based module, our approach leverages both deep learning and quantitative statistical measures. Each component brings unique strengths: the classifier identifies nuanced stylistic differences from labeled data, while perplexity offers an independent gauge of text predictability.

While challenges remain, such as false positives, inherent biases, and the need for transparency, the future of AI text detection is promising. Emerging approaches like multi-modal detection, resilient watermarking, adaptive learning, and explainable AI offer a path

Pluralsight

integrity of written content rather than compromising it.

Axel S.

Axel Sirota is a Microsoft Certified Trainer with a deep interest in Deep Learning and Machine Learning Operations. He has a Masters degree in Mathematics and after researching in Probability, Statistics and Machine Learning optimization, he works as an AI and Cloud Consultant as well as being an Author and Instructor at Pluralsight, Develop Intelligence, and O'Reilly Media.

More about this author

Related Pages

Learn with Hands-on Practice in Real Environments
Business Pricing for Tech Courses
Technology skills for enterprise

Community
Guides
Teach
Partner with Pluralsight
Affiliate Partners
Pluralsight One
Authors

Company
About Us
Careers
Newsroom
Resources

Industries
Education
Financial Services (FSBI)
Healthcare
Insurance
Non-Profit
Public Sector

Get tech insights and upc
Don't miss the latest industry ne
resources, offers, and more.
Sign up now

Pluralsight