# Undergraduate AI Capstone Project #1

111550114 趙堉安

**GitHub Repository URL:** https://github.com/yuan0404/AI-Capstone_Project-1

## Introduction

With the rise of AI-generated content, distinguishing between human-written and AI-generated text has become an increasingly important challenge. This report aims to address this issue by constructing and analyzing a dataset specifically designed to differentiate human-authored text from AI-generated text.

## Dataset Documentation

### 1. Data Description

The dataset used in this study consists of two types of text: human-written content and AI-generated content. The data was collected from publicly available sources and synthetic generation techniques. The human-written content was collected from Wikipedia and the AI-generated content was created using the Llama3.2 model.

The dataset is balanced, containing 193 entries of human-written content and 193 entries of AI-generated content, ensuring that both classes have equal representation.

### 2. Data Collection Process

The data collection process began by retrieving relevant topics from the "Artificial Intelligence" category on Wikipedia using Wikipedia API. Each topic title was extracted, removing special characters, and stored in a JSON file to serve as the basis for both human-written and AI-generated content.

For human-written content, the Wikipedia API was used to fetch the introductory section of each article corresponding to the selected topics. These introductions were extracted as plain text, cleaned by removing extra spaces, and saved as individual TXT files in a designated "human" folder.

For AI-generated content, the same set of topics was used as prompts for the Llama3.2 model. The model was deployed locally using the ollama framework, and each topic was provided as input with an instruction to generate a formal and objective introduction in the style of a Wikipedia article. The generated text was then cleaned and stored in TXT files within an "ai" folder, ensuring a consistent structure with the human-written data.

Once both types of text were collected, they were combined into a structured dataset. Each entry in the dataset included a text sample and a label (human or ai). The final dataset was saved as a CSV file named "dataset.csv", which contained two columns: "text" and "label", ensuring equal representation of human-written and AI-generated content.

3. **Example Topic: Perusall**

**Human-Written Text Example**

Perusall is a social web annotation tool intended for use by students at schools and universities. It allows users to annotate the margins of a text in a virtual group setting that is similar to social media—with upvoting, emojis, chat functionality, and notification. It also includes automatic AI grading.

**AI-Generated Text Example**

Perusall is an online annotation and discussion platform designed to facilitate collaborative learning and academic engagement. Developed by Perusall LLC, a US-based company, this platform allows students, educators, and researchers to annotate and discuss educational materials, such as e-books, articles, and multimedia content, in real-time.

## Algorithms

1. **Feature Extraction**

To effectively distinguish between human-written and AI-generated text, we primarily used BERT embeddings for feature extraction while also preparing Word2Vec embeddings for comparison in later analysis.

BERT captures contextual meaning by considering both preceding and following words. We used the pre-trained "bert-base-uncased" model to extract text representations, where each sample was tokenized and processed, with the final-layer [CLS] token used as the feature vector.

We also used Word2Vec embeddings for comparison, which capture semantic relationships based on word co-occurrence. Tokenized text was converted into numerical representations by averaging word vectors to obtain overall text features.

The extracted BERT and Word2Vec embeddings were organized into feature sets, serving as inputs for classification and clustering models.

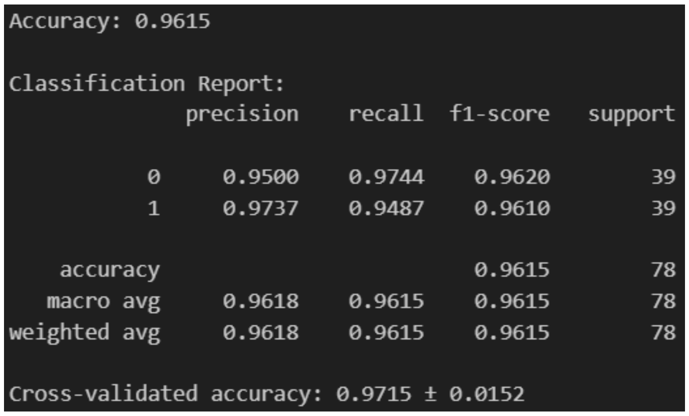
2. **Supervised Learning**

**Logistic Regression**

Logistic Regression models the probability of a given input belonging to a particular class using a logistic function, which outputs a value between 0 and 1. The model assumes a linear relationship between the features and the target variable.

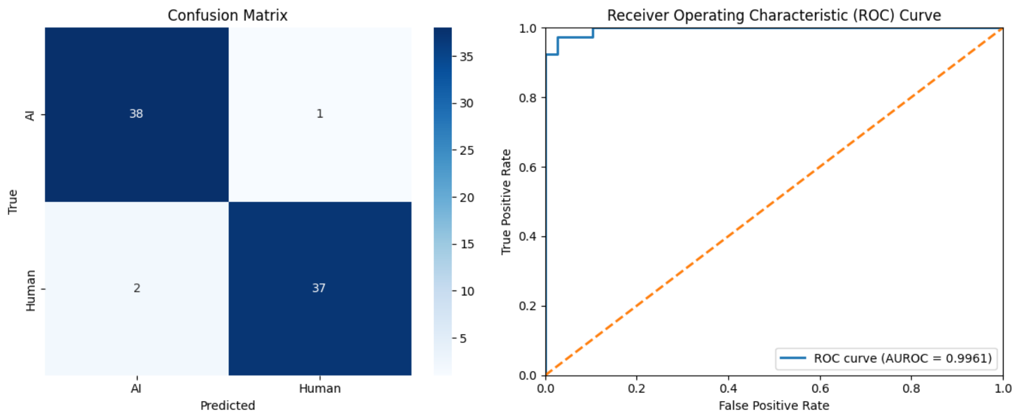
We used the Logistic Regression implementation from the scikit-learn library, with a maximum number 1000 of iterations to ensure convergence during training. Also, we employed stratified splitting, ensuring that the proportions were preserved in both the training and test sets. After training, we used several metrics to evaluate the model, including:

- Accuracy, Classification Report, and Cross-Validation



```
Accuracy: 0.9615

Classification Report:
              precision    recall  f1-score   support

           0     0.9500    0.9744    0.9620        39
           1     0.9737    0.9487    0.9610        39

    accuracy                         0.9615        78
   macro avg     0.9618    0.9615    0.9615        78
weighted avg     0.9618    0.9615    0.9615        78

Cross-validated accuracy: 0.9715 ± 0.0152
```

- Confusion Matrix and ROC curve

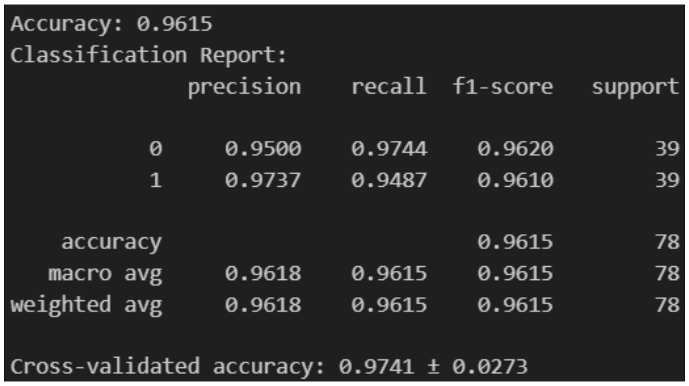


## Support Vector Machines (SVM)

SVM aims to find an optimal hyperplane that separates the data into two distinct classes. It is particularly effective in cases where the data is not linearly separable by transforming it into a higher-dimensional space using a kernel function.

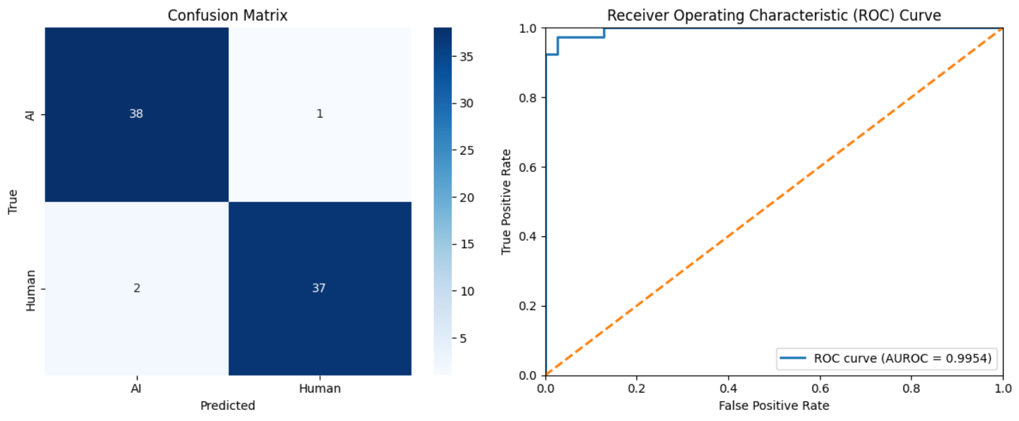
We used the Support Vector Classification (SVC) implementation from the scikit-learn library, setting a linear kernel and the regularization parameter C=1.0. The model was trained on the same data split as logistic regression, ensuring a fair comparison. After training, we used the same evaluation metrics as for logistic regression, including:

- Accuracy, Classification Report, and Cross-Validation



```
Accuracy: 0.9615
Classification Report:
              precision    recall  f1-score   support

           0     0.9500    0.9744    0.9620        39
           1     0.9737    0.9487    0.9610        39

    accuracy                         0.9615        78
   macro avg     0.9618    0.9615    0.9615        78
weighted avg     0.9618    0.9615    0.9615        78

Cross-validated accuracy: 0.9741 ± 0.0273
```

- Confusion Matrix and ROC curve



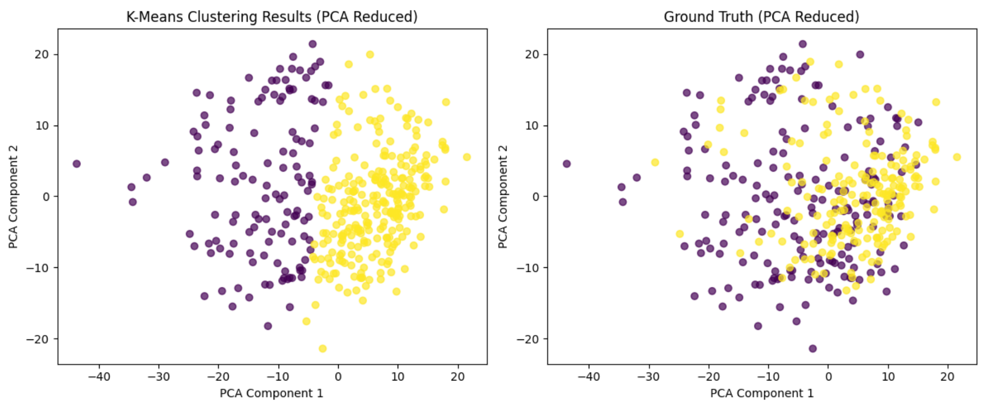
## 3. Unsupervised Learning

### K-Means Clustering

K-Means Clustering is used to partition data into clusters. It iteratively assigns each data point to the cluster with the nearest means and then recalculates the cluster centroids. The process repeats until the convergence.

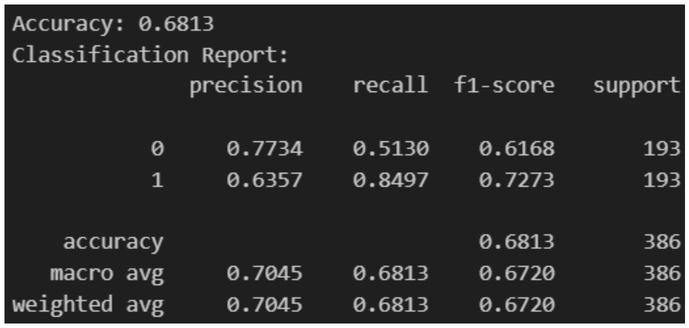
We used the K-Means algorithm and Standard Scaler from the scikit-learn library and applied Principal Component Analysis (PCA) for visualization purposes. PCA is a dimensionality reduction technique that projects data into a lower-dimensional space while preserving as much variance as possible.

This is a comparison of the distribution of predicted data labels and actual labels:



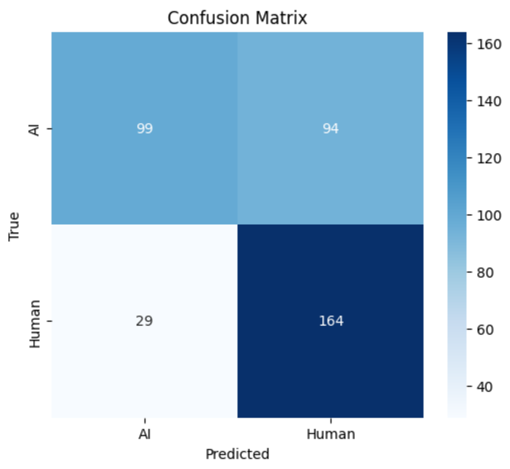
After training, we used the following evaluation metrics with labels as external evaluation metrics, including:

- Accuracy and Classification Report



```
Accuracy: 0.6813
Classification Report:
              precision    recall  f1-score   support

           0     0.7734    0.5130    0.6168       193
           1     0.6357    0.8497    0.7273       193

    accuracy                         0.6813       386
   macro avg     0.7045    0.6813    0.6720       386
weighted avg     0.7045    0.6813    0.6720       386
```

- Confusion Matrix



## Experiments & Analysis: Feature Engineering

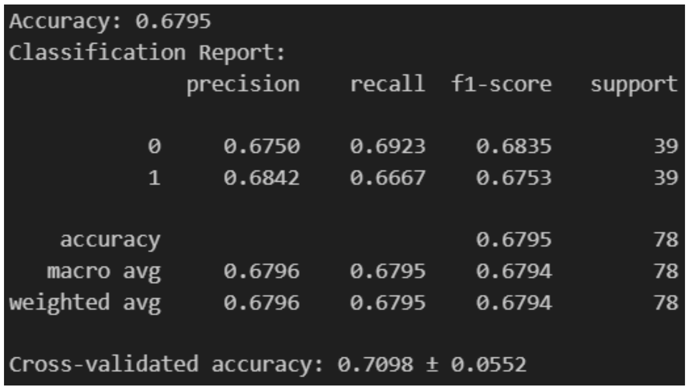
We compare BERT and Word2Vec embeddings to assess their impact on text classification performance, focusing on how contextual and non-contextual embeddings affect model performance and generalization.

### 1. Supervised Learning

#### Logistic Regression

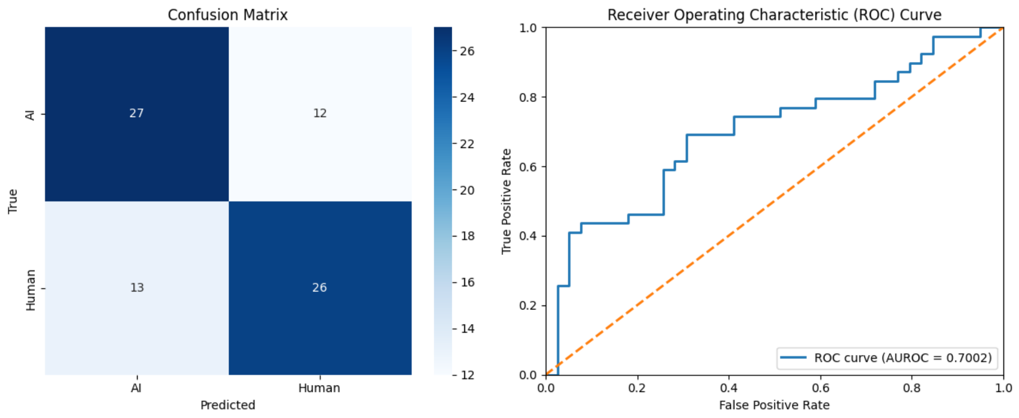
Repeat the previous steps using BERT, but this time, we use Word2Vec for feature extraction. We then compare the results using the same evaluation metrics as before, including:

- Accuracy, Classification Report, and Cross-Validation



```
Accuracy: 0.6795
Classification Report:
              precision    recall  f1-score   support

           0     0.6750    0.6923    0.6835        39
           1     0.6842    0.6667    0.6753        39

    accuracy                         0.6795        78
   macro avg     0.6796    0.6795    0.6794        78
weighted avg     0.6796    0.6795    0.6794        78

Cross-validated accuracy: 0.7098 ± 0.0552
```

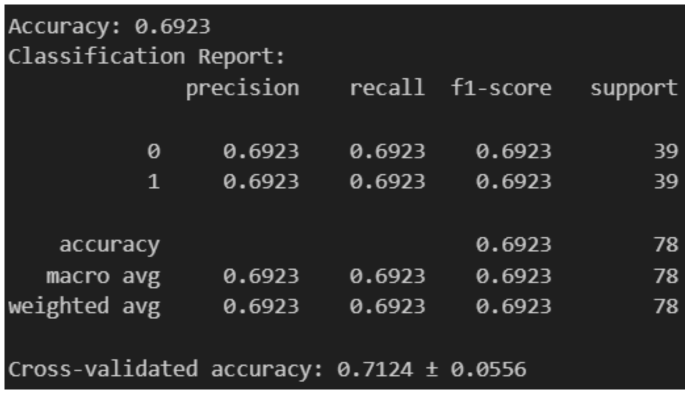
- Confusion Matrix and ROC curve



## Support Vector Machines (SVM)

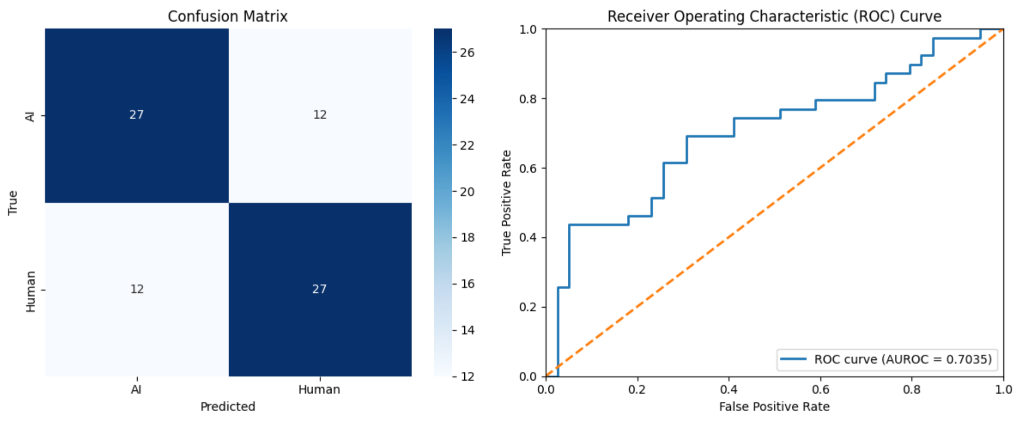
We use Word2Vec for feature extraction as for logistic regression. We then compare the results using the same evaluation metrics as before, including:

- Accuracy, Classification Report, and Cross-Validation



```
Accuracy: 0.6923
Classification Report:
              precision    recall  f1-score   support

           0     0.6923    0.6923    0.6923        39
           1     0.6923    0.6923    0.6923        39

    accuracy                         0.6923        78
   macro avg     0.6923    0.6923    0.6923        78
weighted avg     0.6923    0.6923    0.6923        78

Cross-validated accuracy: 0.7124 ± 0.0556
```

- Confusion Matrix and ROC curve



## Analysis

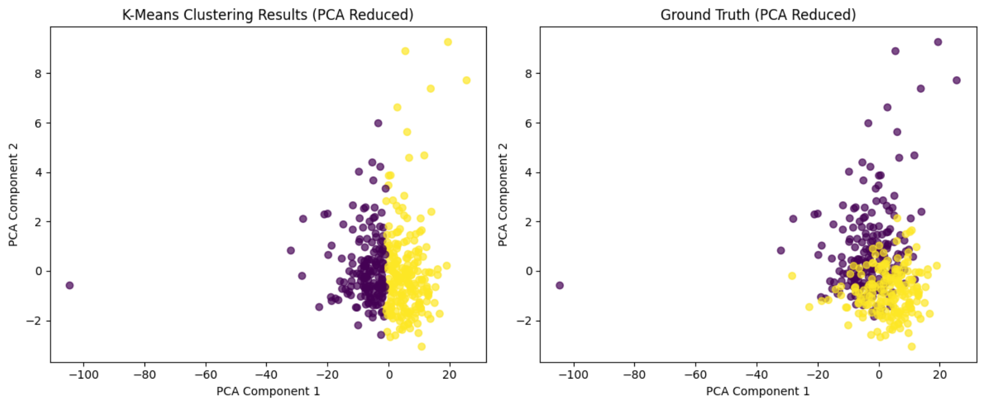
BERT features outperform Word2Vec in text classification due to BERT's ability to capture deep contextual information, while Word2Vec is limited by its independent word representation. Logistic Regression and SVM perform similarly with BERT, but SVM is slightly more stable in cross-validation, likely due to its better adaptation to high-dimensional data.

BERT features significantly enhance classification accuracy, yielding good results for both Logistic Regression and SVM. However, when the feature representation is limited, both models' performance decline, emphasizing that feature selection.

## 2. Unsupervised Learning Feature Engineering

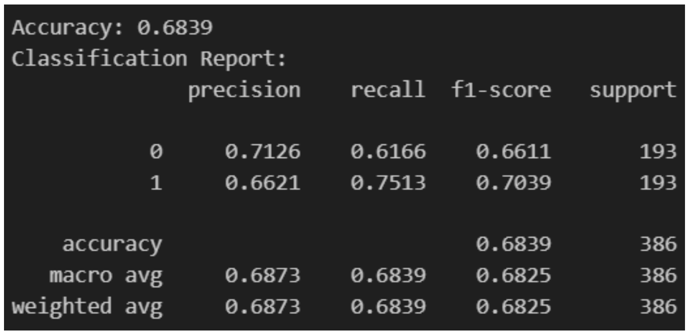
### K-Means Clustering

We use Word2Vec for feature extraction here. This is a comparison of the distribution of predicted data labels and actual labels using Word2Vec embeddings:



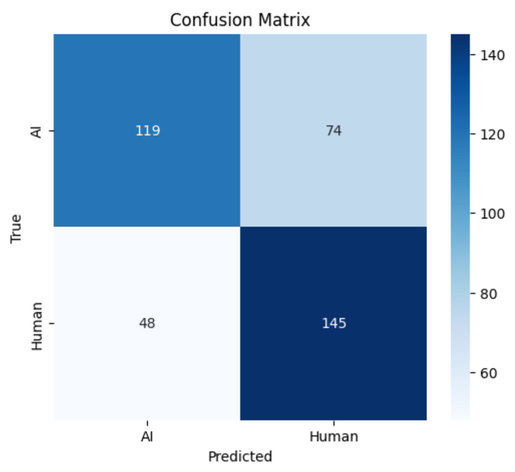
We then compare the results using the same evaluation metrics as before, including:

- Accuracy and Classification Report



```
Accuracy: 0.6839
Classification Report:
              precision    recall  f1-score   support

           0     0.7126    0.6166    0.6611       193
           1     0.6621    0.7513    0.7039       193

    accuracy                         0.6839       386
   macro avg     0.6873    0.6839    0.6825       386
weighted avg     0.6873    0.6839    0.6825       386
```

- Confusion Matrix



### Analysis

BERT and Word2Vec perform similarly. Although BERT has higher precision in some categories, its recall rate is lower, while Word2Vec shows a better balance between precision and recall. Overall, Word2Vec slightly outperforms BERT in terms of balance and classification effectiveness.

## Discussion

The results matched expectations, with BERT outperforming Word2Vec in classification due to its deep contextual understanding, while Word2Vec provided more balanced precision and recall. Performance was influenced by dataset size, balance, and feature representation. A larger dataset or additional preprocessing could improve results.

With more time, fine-tuning BERT, testing other embeddings like GloVe, and experimenting with different classification models would be beneficial. The experiments emphasized the importance of feature selection, but questions remain on generalization to other text domains and the impact of diverse datasets.

## Reference

1. **Human-Written Data:** Category: Artificial intelligence - Wikipedia
2. **AI-Generated Data:** llama3.2

## Import Libraries

```
In [1]: import os
        import json
        import re
        import requests
        from tqdm import tqdm
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import torch
        from transformers import BertTokenizer, BertModel
        from gensim.models import Word2Vec
        from nltk.tokenize import word_tokenize
        from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC
        from sklearn.cluster import KMeans
        from sklearn.metrics import accuracy_score, classification_report, confusion_mat
        from sklearn.model_selection import train_test_split, cross_val_score
        from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA
```

## Create Dataset

### Fetch Topics

```
In [2]: WIKI_API_URL = "https://en.wikipedia.org/w/api.php"
        TOPICS_FILE = "topics.json"

        category_title = "Category:Artificial_intelligence"
        all_pages = []
        params = {
            "action": "query",
            "format": "json",
            "list": "categorymembers",
            "cmtitle": category_title,
            "cmlimit": "max"
        }

        while True:
            response = requests.get(WIKI_API_URL, params=params)
            data = response.json()
            pages = data.get("query", {}).get("categorymembers", [])

            for page in pages:
                title = page["title"]

                if ":" not in title:
                    all_pages.append(title)

            if "continue" in data:
                params["cmcontinue"] = data["continue"]["cmcontinue"]
```

```
    else:
        break

with open(TOPICS_FILE, "w", encoding="utf-8") as f:
    json.dump(all_pages, f, ensure_ascii=False, indent=4)

print(f"Topics have been saved to {TOPICS_FILE}")
```

```
Topics have been saved to topics.json
```

## Fetch Human-written Content

In [3]:
```
WIKI_API_URL = "https://en.wikipedia.org/w/api.php"
TOPICS_FILE = "topics.json"
HUMAN_FOLDER = "human"
os.makedirs(HUMAN_FOLDER, exist_ok=True)

topics = []

with open(TOPICS_FILE, "r", encoding="utf-8") as f:
    topics = json.load(f)

updated_topics = []

for topic in tqdm(topics, desc="Fetching", unit="topic"):
    params = {
        "action": "query",
        "format": "json",
        "titles": topic,
        "prop": "extracts",
        "exintro": True,
        "explaintext": True
    }
    response = requests.get(WIKI_API_URL, params=params)
    data = response.json()
    page = next(iter(data["query"]["pages"].values()), None)

    if page and "extract" in page:
        content = page["extract"]
        cleaned_topic = re.sub(r'[<>:"/\\|?*]', '', topic)
        filename = os.path.join(HUMAN_FOLDER, f"{cleaned_topic}.txt")

        with open(filename, "w", encoding="utf-8") as f:
            f.write(content)

        updated_topics.append(topic)

with open(TOPICS_FILE, "w", encoding="utf-8") as f:
    json.dump(updated_topics, f, ensure_ascii=False, indent=4)
```

```
Fetching: 100%|██████████| 193/193 [01:24<00:00,  2.28topic/s]
```

## Generate AI-written Content

In [4]:
```
OLLAMA_URL = "http://localhost:11434/api/generate"
MODEL = "llama3.2"
TOPICS_FILE = "topics.json"
AI_FOLDER = "ai"
```

```
os.makedirs(AI_FOLDER, exist_ok=True)

topics = []

with open(TOPICS_FILE, "r", encoding="utf-8") as f:
    topics = json.load(f)

updated_topics = []

for topic in tqdm(topics, desc="Fetching", unit="topic"):
    content = []

    prompt = f"Write an introduction in the style of a Wikipedia article for the
    payload = {"model": MODEL, "prompt": prompt, "stream": False}
    response = requests.post(OLLAMA_URL, json=payload)

    if response.status_code == 200:
        content = response.json().get("response", "")

    if content:
        cleaned_topic = re.sub(r'[<>:"/\\|?*]', '', topic)
        filename = os.path.join(AI_FOLDER, f"{cleaned_topic}.txt")

        with open(filename, "w", encoding="utf-8") as f:
            f.write(content)

        updated_topics.append(topic)

with open(TOPICS_FILE, "w", encoding="utf-8") as f:
    json.dump(updated_topics, f, ensure_ascii=False, indent=4)
```

```
Fetching: 100%|██████████| 193/193 [43:01<00:00, 13.38s/topic]
```

## Create Dataset

In [5]:

```
AI_FOLDER = "ai"
HUMAN_FOLDER = "human"
DATASET_FILE = "dataset.csv"

def load_data_from_folder(folder, label):
    data = []

    for filename in os.listdir(folder):
        filepath = os.path.join(folder, filename)

        with open(filepath, 'r', encoding='utf-8') as file:
            text = file.read()
            cleaned_text = text.replace('\n', ' ').replace('\r', '')
            cleaned_text = re.sub(r'\s+', ' ', cleaned_text)
            data.append({'text': cleaned_text, 'label': label})

    return data

human_data = load_data_from_folder(HUMAN_FOLDER, 'human')
ai_data = load_data_from_folder(AI_FOLDER, 'ai')
dataset = pd.DataFrame(human_data + ai_data)
dataset.to_csv(DATASET_FILE, index=False, encoding='utf-8')
```

```python
print(f"Dataset has been saved to {DATASET_FILE}")
```

```
Dataset has been saved to dataset.csv
```

# Data Preprocessing

## Feature Extraction

In [6]:
```python
def get_bert_features(text):
    encoded = tokenizer(text,
                        add_special_tokens=True,
                        truncation=True,
                        padding=True,
                        max_length=512,
                        return_tensors='pt')

    with torch.no_grad():
        outputs = model(**encoded)

    cls_vector = outputs.last_hidden_state[0][0].numpy()

    return cls_vector


def get_text_features(text):
    vectors = [model.wv[word] for word in text if word in model.wv]

    if vectors:
        return sum(vectors) / len(vectors)
    else:
        return [0] * model.vector_size


df = pd.read_csv('dataset.csv')
df['text'] = df['text'].fillna("").astype(str)

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
df['bert_vector'] = df['text'].apply(get_bert_features)

df['tokens'] = df['text'].apply(lambda x: word_tokenize(x.lower()))
model = Word2Vec(sentences=df['tokens'], vector_size=100, window=5, min_count=1,
df['word_vector'] = df['tokens'].apply(get_text_features)
```

## Data Preparing

In [7]:
```python
label_mapping = {
    'human': 0,
    'ai': 1
}

df['label'] = df['label'].map(label_mapping)

X_bert = list(df['bert_vector'])
```

```
X_word = list(df['word_vector'])

y = df['label']
```

# Supervised Learning

## Model Training

### Logistic Regression

In [8]:

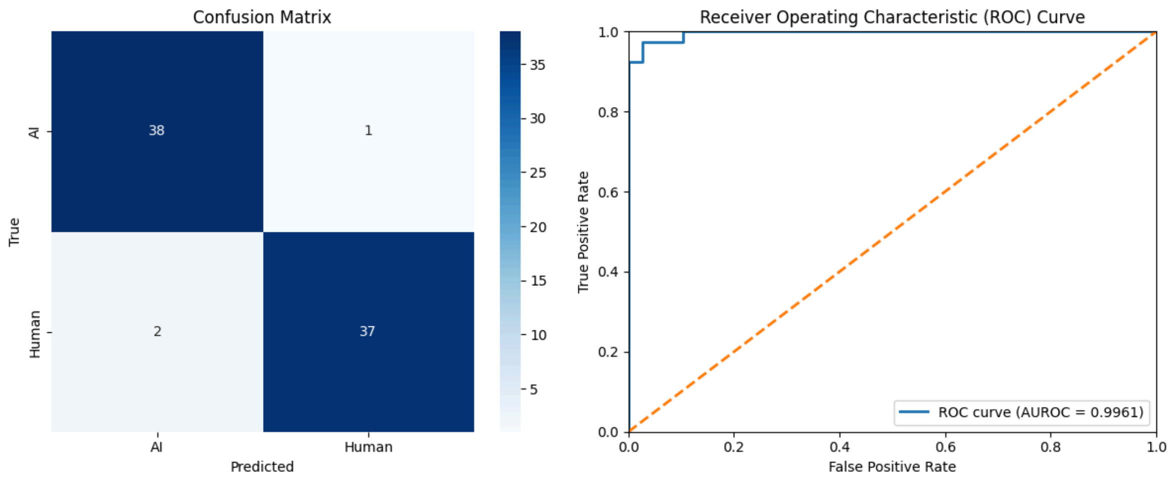
```
X_train, X_test, y_train, y_test = train_test_split(X_bert, y, test_size=0.2, ra

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}\n")
print(f"Classification Report:\n{classification_report(y_test, y_pred, digits=4)

cv_scores = cross_val_score(model, X_bert, y, cv=5)
print(f"Cross-validated accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}
```

```
Accuracy: 0.9615

Classification Report:
              precision    recall  f1-score   support

           0     0.9500    0.9744    0.9620        39
           1     0.9737    0.9487    0.9610        39

    accuracy                         0.9615        78
   macro avg     0.9618    0.9615    0.9615        78
weighted avg     0.9618    0.9615    0.9615        78

Cross-validated accuracy: 0.9715 ± 0.0152
```

In [9]:

```
cm = confusion_matrix(y_test, y_pred)

y_pred_prob = model.predict_proba(X_test)[:, 1]
auroc = roc_auc_score(y_test, y_pred_prob)
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['AI', 'Human'],
axes[0].set_title('Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('True')

axes[1].plot(fpr, tpr, lw=2, label=f'ROC curve (AUROC = {auroc:.4f})')
axes[1].plot([0, 1], [0, 1], lw=2, linestyle='--')
axes[1].set_xlim([0.0, 1.0])
axes[1].set_ylim([0.0, 1.0])
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Receiver Operating Characteristic (ROC) Curve')
```

```
axes[1].legend(loc="lower right")

plt.tight_layout()
plt.show()
```



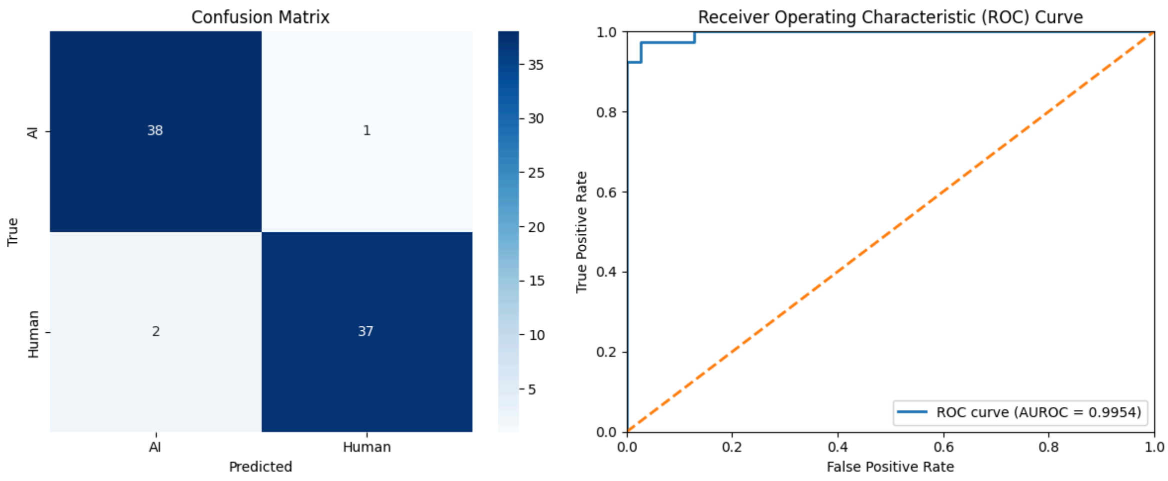
## Support Vector Machine (SVM)

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X_bert, y, test_size=0.2, ra

model = SVC(kernel='linear', C=1.0, random_state=42, probability=True)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Classification Report:\n{classification_report(y_test, y_pred, digits=4)

cv_scores = cross_val_score(model, X_bert, y, cv=5)
print(f"Cross-validated accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}
```

```
Accuracy: 0.9615
Classification Report:
              precision    recall  f1-score   support

           0     0.9500    0.9744    0.9620        39
           1     0.9737    0.9487    0.9610        39

    accuracy                         0.9615        78
   macro avg     0.9618    0.9615    0.9615        78
weighted avg     0.9618    0.9615    0.9615        78

Cross-validated accuracy: 0.9741 ± 0.0273
```

In [11]:
```
cm = confusion_matrix(y_test, y_pred)

y_pred_prob = model.predict_proba(X_test)[:, 1]
auroc = roc_auc_score(y_test, y_pred_prob)
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['AI', 'Human'],
axes[0].set_title('Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('True')
```

```
axes[1].plot(fpr, tpr, lw=2, label=f'ROC curve (AUROC = {auroc:.4f})')
axes[1].plot([0, 1], [0, 1], lw=2, linestyle='--')
axes[1].set_xlim([0.0, 1.0])
axes[1].set_ylim([0.0, 1.0])
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Receiver Operating Characteristic (ROC) Curve')
axes[1].legend(loc="lower right")

plt.tight_layout()
plt.show()
```



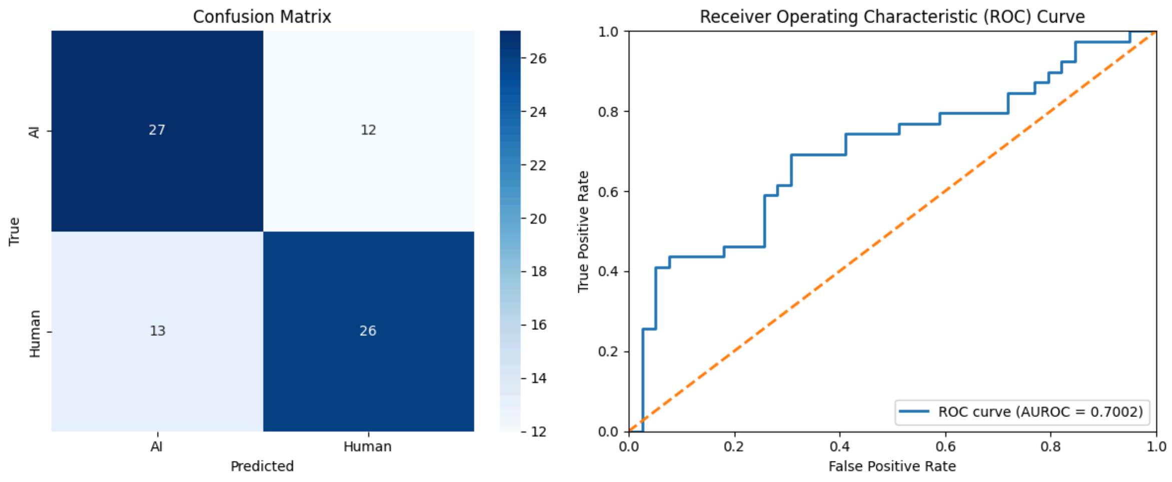
## Feature Engineering

### Logistic Regression

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(X_word, y, test_size=0.2, ra

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Classification Report:\n{classification_report(y_test, y_pred, digits=4)

cv_scores = cross_val_score(model, X_word, y, cv=5)
print(f"Cross-validated accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}
```

```
Accuracy: 0.6795
Classification Report:
              precision    recall  f1-score   support

           0     0.6750    0.6923    0.6835        39
           1     0.6842    0.6667    0.6753        39

    accuracy                         0.6795        78
   macro avg     0.6796    0.6795    0.6794        78
weighted avg     0.6796    0.6795    0.6794        78

Cross-validated accuracy: 0.7098 ± 0.0552
```

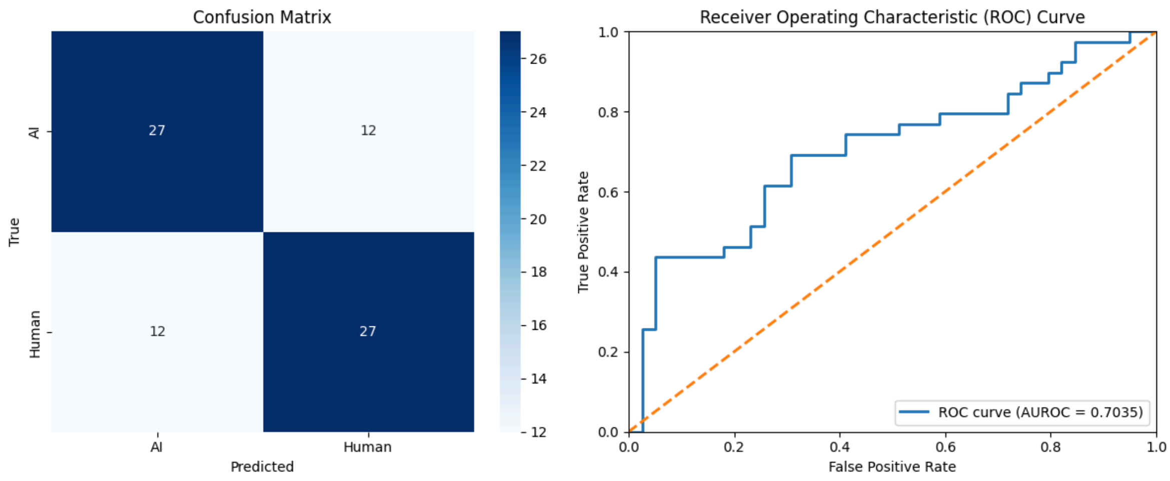
In [13]:
```
cm = confusion_matrix(y_test, y_pred)

y_pred_prob = model.predict_proba(X_test)[:, 1]
auroc = roc_auc_score(y_test, y_pred_prob)
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['AI', 'Human'],
axes[0].set_title('Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('True')

axes[1].plot(fpr, tpr, lw=2, label=f'ROC curve (AUROC = {auroc:.4f})')
axes[1].plot([0, 1], [0, 1], lw=2, linestyle='--')
axes[1].set_xlim([0.0, 1.0])
axes[1].set_ylim([0.0, 1.0])
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Receiver Operating Characteristic (ROC) Curve')
axes[1].legend(loc="lower right")

plt.tight_layout()
plt.show()
```



## Support Vector Machine (SVM)

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X_word, y, test_size=0.2, ra

model = SVC(kernel='linear', C=1.0, random_state=42, probability=True)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Classification Report:\n{classification_report(y_test, y_pred, digits=4)

cv_scores = cross_val_score(model, X_word, y, cv=5)
print(f"Cross-validated accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}
Accuracy: 0.6923
Classification Report:
              precision    recall  f1-score   support

           0     0.6923    0.6923    0.6923        39
           1     0.6923    0.6923    0.6923        39

    accuracy                         0.6923        78
   macro avg     0.6923    0.6923    0.6923        78
weighted avg     0.6923    0.6923    0.6923        78

Cross-validated accuracy: 0.7124 ± 0.0556
```

In [15]:
```python
cm = confusion_matrix(y_test, y_pred)

y_pred_prob = model.predict_proba(X_test)[:, 1]
auroc = roc_auc_score(y_test, y_pred_prob)
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['AI', 'Human'],
axes[0].set_title('Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('True')

axes[1].plot(fpr, tpr, lw=2, label=f'ROC curve (AUROC = {auroc:.4f})')
axes[1].plot([0, 1], [0, 1], lw=2, linestyle='--')
axes[1].set_xlim([0.0, 1.0])
axes[1].set_ylim([0.0, 1.0])
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Receiver Operating Characteristic (ROC) Curve')
axes[1].legend(loc="lower right")

plt.tight_layout()
plt.show()
```

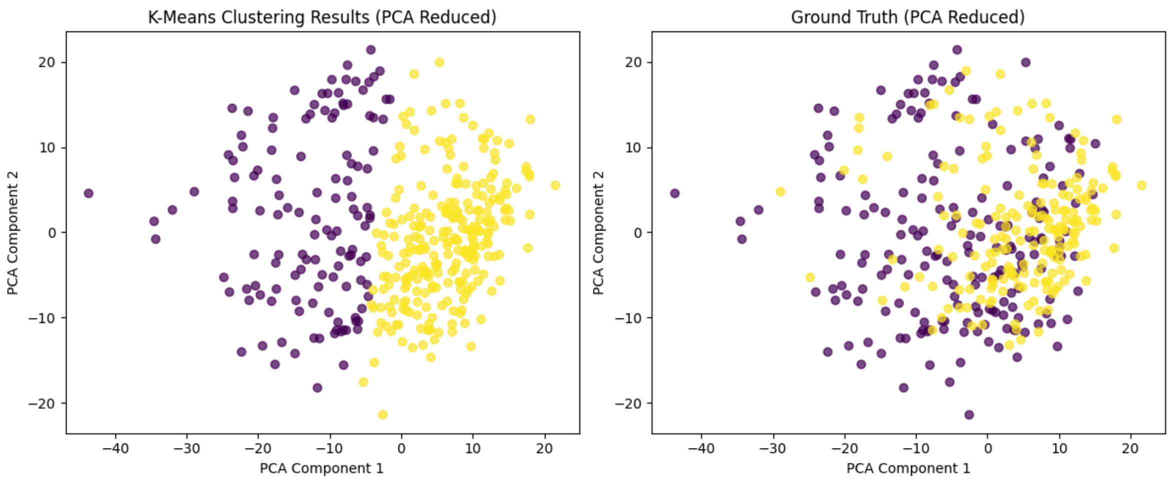


# Unsupervised Learning

## Model Training

## K-Means Clustering

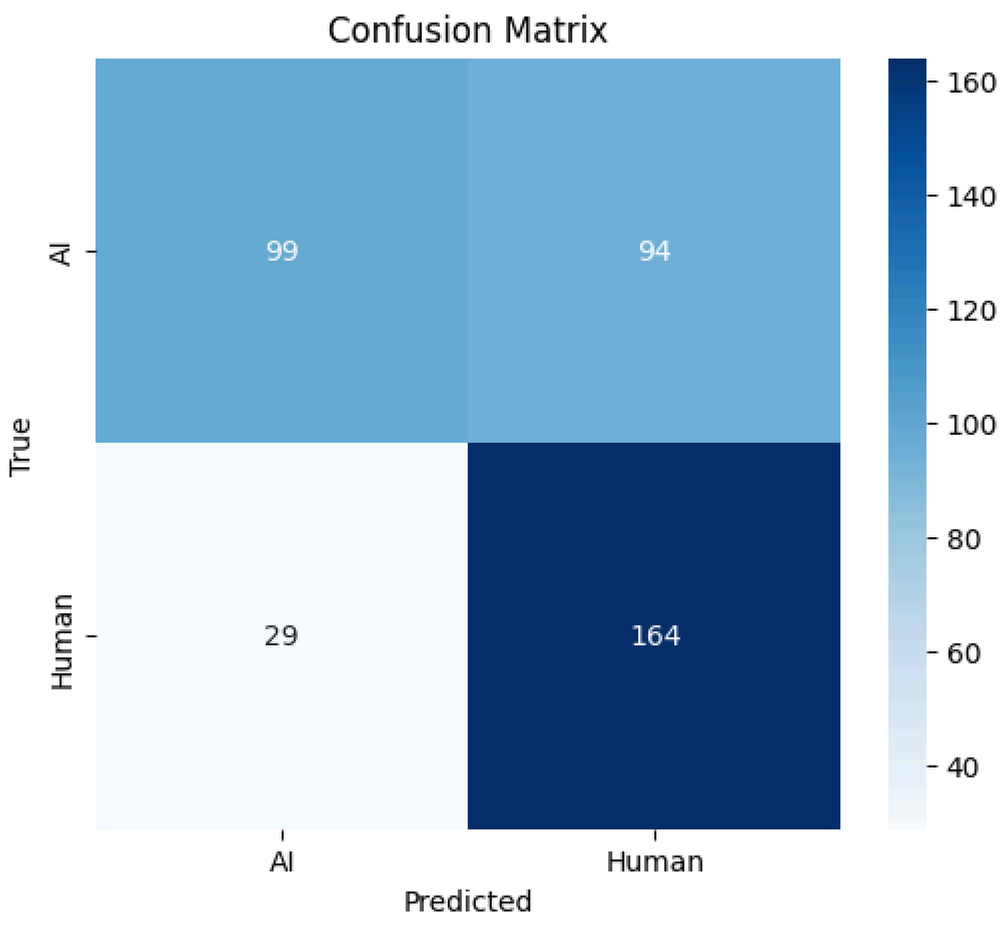
In [20]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_bert)

model = KMeans(n_clusters=2, random_state=42)
y_pred = model.fit_predict(X_scaled)

reducer = PCA(n_components=2)
X_reduced = reducer.fit_transform(X_scaled)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

scatter1 = axes[0].scatter(X_reduced[:, 0], X_reduced[:, 1], c=y_pred, cmap='vir
axes[0].set_title("K-Means Clustering Results (PCA Reduced)")
axes[0].set_xlabel("PCA Component 1")
axes[0].set_ylabel("PCA Component 2")

scatter2 = axes[1].scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis'
axes[1].set_title("Ground Truth (PCA Reduced)")
axes[1].set_xlabel("PCA Component 1")
axes[1].set_ylabel("PCA Component 2")

plt.tight_layout()
plt.show()
```



In [21]:
```
print(f"Accuracy: {accuracy_score(y, y_pred):.4f}")
print(f"Classification Report:\n{classification_report(y, y_pred, digits=4)}")
```

```
Accuracy: 0.6813
Classification Report:
              precision    recall  f1-score   support

           0     0.7734    0.5130    0.6168       193
           1     0.6357    0.8497    0.7273       193

    accuracy                         0.6813       386
   macro avg     0.7045    0.6813    0.6720       386
weighted avg     0.7045    0.6813    0.6720       386
```

In [22]:
```
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['AI', 'Human'],
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

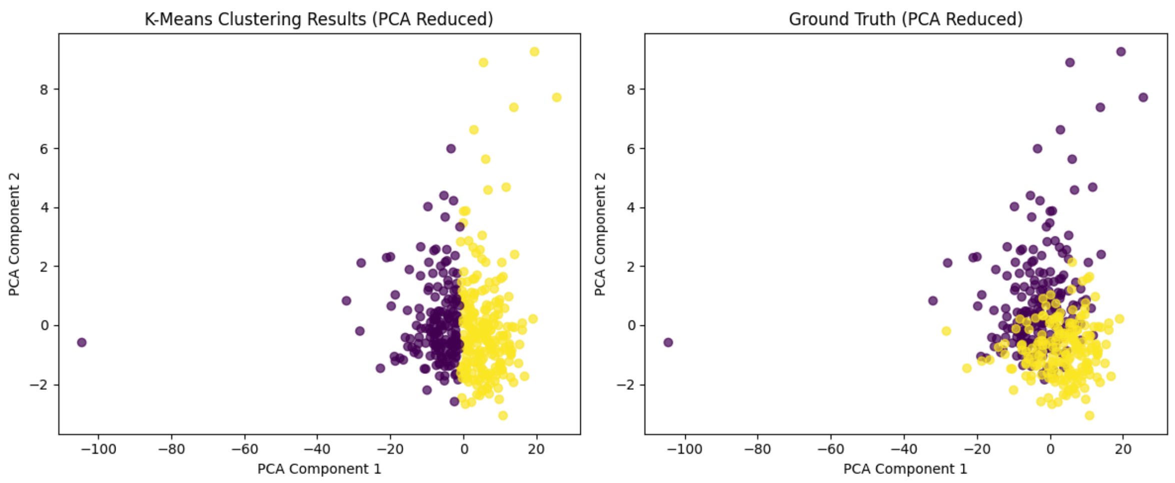


## Feature Engineering

### K-Means Clustering

In [23]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_word)

model = KMeans(n_clusters=2, random_state=42)
y_pred = model.fit_predict(X_scaled)

reducer = PCA(n_components=2)
X_reduced = reducer.fit_transform(X_scaled)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

scatter1 = axes[0].scatter(X_reduced[:, 0], X_reduced[:, 1], c=y_pred, cmap='vir
axes[0].set_title("K-Means Clustering Results (PCA Reduced)")
axes[0].set_xlabel("PCA Component 1")
axes[0].set_ylabel("PCA Component 2")

scatter2 = axes[1].scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis'
axes[1].set_title("Ground Truth (PCA Reduced)")
axes[1].set_xlabel("PCA Component 1")
axes[1].set_ylabel("PCA Component 2")
```

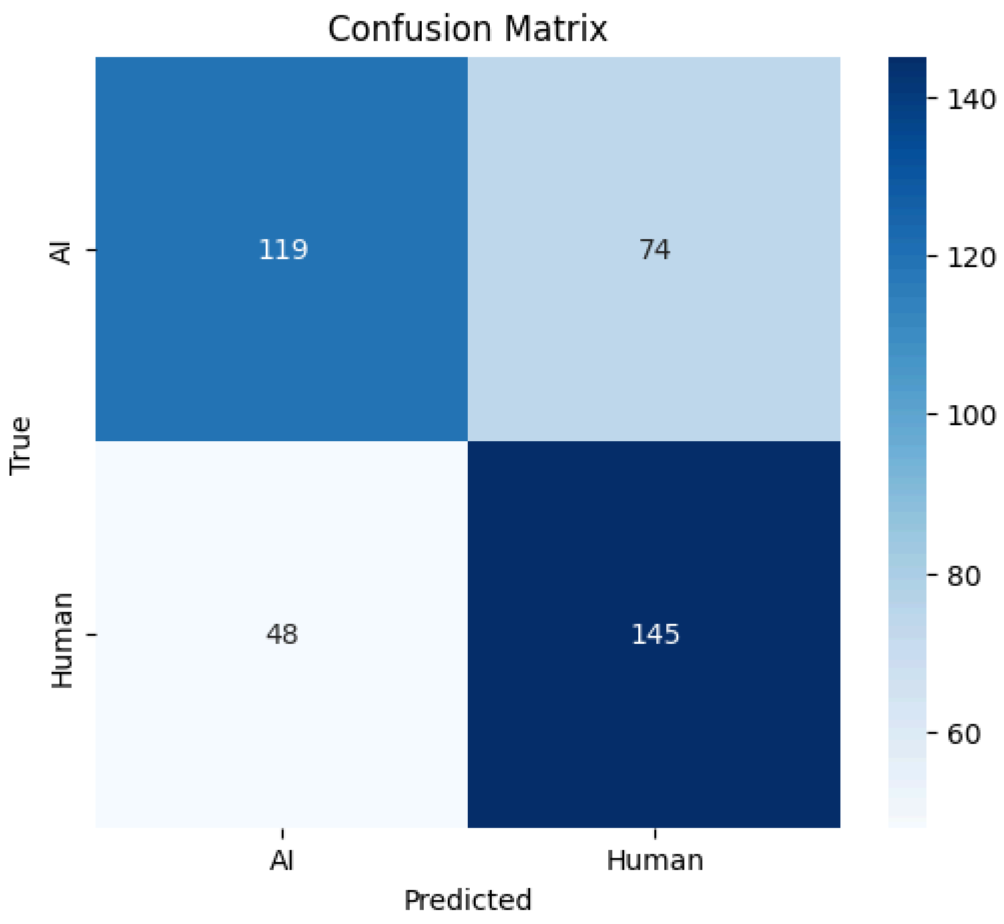
```python
plt.tight_layout()
plt.show()
```



In [24]:
```python
print(f"Accuracy: {accuracy_score(y, y_pred):.4f}")
print(f"Classification Report:\n{classification_report(y, y_pred, digits=4)}")
```

```
Accuracy: 0.6839
Classification Report:
              precision    recall  f1-score   support

           0     0.7126    0.6166    0.6611       193
           1     0.6621    0.7513    0.7039       193

    accuracy                         0.6839       386
   macro avg     0.6873    0.6839    0.6825       386
weighted avg     0.6873    0.6839    0.6825       386
```

In [25]:
```python
cm = confusion_matrix(y, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['AI', 'Human'],
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```


Confusion Matrix
True
Predicted
AI
Human
AI
Human
119
74
48
145
140
120
100
80
60