🧠 Fake News Generator & Detector using Generative AI and NLP

Author: Akash Dhar Dubey

This notebook demonstrates the use of Generative AI to create and detect fake news using GPT-2 and BERT.

🔧 Setup & Install Required Libraries

```
!pip install transformers torch pandas scikit-learn
```

📦 Import Libraries

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer,
BertTokenizer, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
import torch
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

✍️ Fake News Generator using GPT-2

```
gpt2_tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
gpt2_model = GPT2LMHeadModel.from_pretrained("gpt2")

def generate_fake_news(prompt, max_length=50):
    input_ids = gpt2_tokenizer.encode(prompt, return_tensors='pt')
    output = gpt2_model.generate(input_ids, max_length=max_length,
num_return_sequences=1)
    return gpt2_tokenizer.decode(output[0], skip_special_tokens=True)

# Example
generate_fake_news("Breaking news:")
```

Fake News Detection using BERT

```
# Sample dataset: Replace this with actual path or dataset
df = pd.DataFrame({
    'text': [
```

```
        "Aliens landed in New York City.",
        "The president gave a speech on economy today.",
        "Scientists discover a new planet.",
        "Actor wins award for performance in thriller movie."
    ],
    'label': [1, 0, 1, 0]  # 1 = Fake, 0 = Real
})

train_texts, val_texts, train_labels, val_labels =
train_test_split(df['text'], df['label'], test_size=0.2)

bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_encodings = bert_tokenizer(list(train_texts), truncation=True,
padding=True, return_tensors="pt")
val_encodings = bert_tokenizer(list(val_texts), truncation=True,
padding=True, return_tensors="pt")

class NewsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
    def __len__(self):
        return len(self.labels)

train_dataset = NewsDataset(train_encodings, list(train_labels))
val_dataset = NewsDataset(val_encodings, list(val_labels))

bert_model = BertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=2)

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=2,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    warmup_steps=10,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=5,
    evaluation_strategy="epoch"
)

trainer = Trainer(
    model=bert_model,
    args=training_args,
    train_dataset=train_dataset,
```

```
    eval_dataset=val_dataset
)

trainer.train()
```

Evaluate BERT Model

```
preds = trainer.predict(val_dataset)
pred_labels = np.argmax(preds.predictions, axis=1)
print(classification_report(val_labels, pred_labels))
```