Exposing the Truth with Advanced Fake News Detection Powered by NLP

Project: Complete Python implementation (transformers + FastAPI + frontend)

```
Project: Exposing the Truth with Advanced Fake News Detection Powered by Natural Language Processin

I assumed you want a practical, modern implementation using Python (transformer-based model), train

Project structure, requirements, training, evaluation, and a simple web API + frontend are included

---

Project scaffold (Python, Hugging Face + FastAPI)

fake-news-nlp/
■■ data/
■  ■■ train.csv        # columns: text,label  (label: 0=real,1=fake)
■  ■■ val.csv
■■ requirements.txt
■■ train.py
■■ eval.py
■■ inference_api.py
■■ utils.py
■■ model/
■  ■■ (saved model files after training)
■■ web/
■  ■■ index.html
■■ README.md

---

requirements.txt

python>=3.9
transformers==4.33.0
datasets==2.10.0
torch>=1.13.0
scikit-learn
fastapi
uvicorn
pydantic
python-multipart
gunicorn
sentencepiece
pandas
numpy

---

utils.py

# utils.py
import re
import pandas as pd

def load_csv(path):
    return pd.read_csv(path)
```

```
def clean_text(text: str) -> str:
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r"http\S+", "", text)
    text = re.sub(r"@\w+", "", text)
    text = re.sub(r"[^a-z0-9\s]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text
```

---

train.py

```
# train.py
import os
import argparse
from datasets import Dataset, load_metric
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trai
import pandas as pd
import numpy as np
from utils import load_csv, clean_text

def prepare_dataset(csv_path, tokenizer, text_col="text", label_col="label", max_length=256):
    df = load_csv(csv_path)
    df[text_col] = df[text_col].apply(clean_text)
    ds = Dataset.from_pandas(df[[text_col, label_col]])
    def tokenize_fn(example):
        return tokenizer(example[text_col], truncation=True, padding='max_length', max_length=max_l
    ds = ds.map(tokenize_fn, batched=True)
    ds = ds.rename_column(text_col, "text")
    ds.set_format(type="torch", columns=["input_ids", "attention_mask", label_col])
    return ds

def compute_metrics(eval_pred):
    metric_acc = load_metric("accuracy")
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    acc = metric_acc.compute(predictions=preds, references=labels)
    from sklearn.metrics import precision_recall_fscore_support
    p, r, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    return {"accuracy": acc["accuracy"], "precision": p, "recall": r, "f1": f1}

def main(args):
    model_name = args.model_name
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    train_ds = prepare_dataset(args.train_csv, tokenizer)
    val_ds = prepare_dataset(args.val_csv, tokenizer)

    model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

    training_args = TrainingArguments(
        output_dir=args.output_dir,
        num_train_epochs=args.epochs,
        per_device_train_batch_size=args.train_bs,
        per_device_eval_batch_size=args.eval_bs,
        evaluation_strategy="epoch",
        save_strategy="epoch",
        logging_strategy="steps",
        logging_steps=100,
        load_best_model_at_end=True,
        metric_for_best_model="f1",
        greater_is_better=True,
        fp16=False
    )
```

```
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_ds,
        eval_dataset=val_ds,
        tokenizer=tokenizer,
        compute_metrics=compute_metrics
    )

    trainer.train()
    trainer.save_model(args.output_dir)
    tokenizer.save_pretrained(args.output_dir)
    print("Training finished. Model saved to", args.output_dir)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--train_csv", type=str, default="data/train.csv")
    parser.add_argument("--val_csv", type=str, default="data/val.csv")
    parser.add_argument("--output_dir", type=str, default="model/fake-news-distilbert")
    parser.add_argument("--model_name", type=str, default="distilbert-base-uncased")
    parser.add_argument("--epochs", type=int, default=3)
    parser.add_argument("--train_bs", type=int, default=16)
    parser.add_argument("--eval_bs", type=int, default=32)
    args = parser.parse_args()
    main(args)

Notes

Use a balanced dataset. If labels are imbalanced, consider class_weight or data augmentation.

distilbert-base-uncased is a fast option. For best results, try roberta-base or domain-specific mod



---

eval.py

# eval.py
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import pandas as pd
from utils import clean_text

def predict_texts(texts, model_path="model/fake-news-distilbert"):
    tokenizer = AutoTokenizer.from_pretrained(model_path)
    model = AutoModelForSequenceClassification.from_pretrained(model_path)
    model.eval()
    inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt")
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    preds = torch.argmax(logits, dim=1).tolist()
    probs = torch.softmax(logits, dim=1).tolist()
    return [{"text":t, "label":int(p), "prob_real": float(prob[0]), "prob_fake": float(prob[1])} fo

if __name__ == "__main__":
    example_texts = [
        "Breaking: Celebrity endorses miracle cure that is unproven.",
        "Government announces new initiative to build parks in city center."
    ]
    res = predict_texts([clean_text(t) for t in example_texts])
    import json
    print(json.dumps(res, indent=2))
---
```

inference_api.py (FastAPI)

```
# inference_api.py
from fastapi import FastAPI, Form
from pydantic import BaseModel
from typing import List
from utils import clean_text
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

MODEL_DIR = "model/fake-news-distilbert"

app = FastAPI(title="Fake News Detection API")

class PredictRequest(BaseModel):
    texts: List[str]

# load model and tokenizer once
tokenizer = AutoTokenizer.from_pretrained(MODEL_DIR)
model = AutoModelForSequenceClassification.from_pretrained(MODEL_DIR)
model.eval()

def predict(texts):
    cleaned = [clean_text(t) for t in texts]
    inputs = tokenizer(cleaned, padding=True, truncation=True, return_tensors="pt")
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    probs = torch.softmax(logits, dim=1).cpu().numpy()
    preds = logits.argmax(dim=1).cpu().numpy().tolist()
    results = []
    for t, p, prob in zip(texts, preds, probs):
        results.append({
            "text": t,
            "pred_label": int(p),  # 0 real, 1 fake
            "prob_real": float(prob[0]),
            "prob_fake": float(prob[1])
        })
    return results

@app.post("/predict")
def api_predict(req: PredictRequest):
    return {"predictions": predict(req.texts)}

@app.get("/")
def root():
    return {"message": "Fake News Detection API. Use POST /predict with JSON {texts: [...]}"}
```

Run the API:

```
uvicorn inference_api:app --host 0.0.0.0 --port 8000 --reload
```

---

simple frontend (web/index.html)

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Fake News Demo</title>
  <style>
    body{font-family: Arial; padding: 20px;}
    textarea{width:100%; height:120px;}
    button{padding:10px 20px; margin-top:10px;}
    .result{margin-top:15px; padding:10px; border:1px solid #ddd;}
  </style>
</head>
<body>
  <h2>Fake News Detection — Demo</h2>
  <textarea id="text">Enter news text here...</textarea><br/>
  <button onclick="predict()">Predict</button>
  <div id="out"></div>

<script>
async function predict(){
  const text = document.getElementById("text").value;
  const res = await fetch("/predict", {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify({texts: [text]})
  });
  const j = await res.json();
  const p = j.predictions[0];
  document.getElementById("out").innerHTML = `
    <div class="result">
      <strong>Label:</strong> ${p.pred_label === 1 ? 'Fake' : 'Real'}<br/>
      <strong>Prob (Real):</strong> ${p.prob_real.toFixed(3)}<br/>
      <strong>Prob (Fake):</strong> ${p.prob_fake.toFixed(3)}<br/>
      <p>${p.text}</p>
    </div>
  `;
}
</script>
</body>
</html>
```

> If you serve index.html from the same server, you can add a route in FastAPI to serve static file

---

README.md (quick run)

```
# Fake News Detection (NLP)

1. Prepare dataset:
   - Place `train.csv` and `val.csv` in `data/`. Each CSV must have columns: `text`, `label` (0=rea

2. Install:
   pip install -r requirements.txt

3. Train:
   python train.py --train_csv data/train.csv --val_csv data/val.csv --output_dir model/fake-news-d

4. Run API:
```

```
   uvicorn inference_api:app --reload --host 0.0.0.0 --port 8000

5. Test:
   POST http://localhost:8000/predict
   Body: {"texts":["some news text here"]}

6. Frontend:
   open web/index.html (and ensure fetch points to correct API url)


---

Tips to improve model performance

Use more labeled data (fact-checking datasets: LIAR, FakeNewsNet, etc.).

Try balanced sampling or class weights when labels are imbalanced.

Use longer max_length for long news articles.

Try ensemble: transformer + classical ML features (source reliability, URL patterns).

Add explainability: SHAP or LIME to show why a text was predicted fake.


---

If you want:

I can adapt this to a JSP/Java backend instead.

Or produce a colab-ready notebook with training and inference steps.

Or provide a sample dataset snippet and a pre-made train.csv example.
```