TF_IDFModel

June 17, 2025

[1]:

```
import pandas as pd
import re
import torch
import numpy as np
from tqdm import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from scipy.sparse import hstack
from transformers import AutoTokenizer, AutoModel
```

[2]:

```
# Load dataset
df = pd.read_csv('fake_job_postings.csv')
```

[3]:

```
# Stopwords list
stopwords = set([
    "a", "an", "the", "and", "or", "but", "if", "while", "with", "without",
    "in", "on", "at", "to", "from", "by", "for", "of", "as", "is", "are", "was",
    "were", "be", "been", "being", "this", "that", "these", "those", "it",␣
  ↪"its",
    "he", "she", "they", "them", "his", "her", "their", "you", "your", "we",␣
  ↪"us"
])
```

[4]:

```
# Clean text
def preprocess_text(text):
    text = text.lower()
    tokens = re.findall(r'\b[a-z]{2,}\b', text)
    return ' '.join([t for t in tokens if t not in stopwords])
```

[5]:

```
# Prepare input text
df['text'] = df[['description', 'requirements', 'company_profile']].fillna('').
  ↪agg(' '.join, axis=1)
df['clean_text'] = df['text'].apply(preprocess_text)
```

[6]:

```python
# Load Hugging Face model and tokenizer
print("Loading transformer model...")
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = AutoModel.from_pretrained("distilbert-base-uncased")
model.eval()
```

```
Loading transformer model…
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
```

```
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

```
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed.
Falling back to regular HTTP download. For better performance, install the
package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but
the 'hf_xet' package is not installed. Falling back to regular HTTP download.
For better performance, install the package with: `pip install
huggingface_hub[hf_xet]` or `pip install hf_xet`
```

```
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

[6]:

```
DistilBertModel(
  (embeddings): Embeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (transformer): Transformer(
    (layer): ModuleList(
      (0-5): 6 x TransformerBlock(
        (attention): DistilBertSdpaAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
      )
    )
  )
)
```

[7]:

```python
# Function to get mean pooled embedding
def get_embedding(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True,␣
 ↪padding=True, max_length=128)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
```

[8]:

```python
# Generate embeddings for each job posting
print("Generating embeddings...")
embeddings = np.vstack([get_embedding(text) for text in tqdm(df['clean_text'])])
```

```
Generating embeddings…
```

```
100%|          | 17880/17880 [1:19:20<00:00,  3.76it/s]
```

[9]:

```python
# TF-IDF feature extraction
print("Extracting TF-IDF features...")
tfidf_vectorizer = TfidfVectorizer(max_features=1000)
tfidf_features = tfidf_vectorizer.fit_transform(df['clean_text'])
```

```
Extracting TF-IDF features…
```

[10]:

```python
# LDA topic modeling
print("Running LDA topic modeling...")
count_vectorizer = CountVectorizer(max_features=1000)
count_matrix = count_vectorizer.fit_transform(df['clean_text'])
lda_model = LatentDirichletAllocation(n_components=10, random_state=42)
lda_topics = lda_model.fit_transform(count_matrix)
```

```
Running LDA topic modeling…
```

[11]:

```
# Combine features
print("Combining features...")
X_combined = hstack([tfidf_features, lda_topics, embeddings])
y = df['fraudulent']
```

```
Combining features…
```

[12]:

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, stratify=y,␣
 ↪test_size=0.2, random_state=42)
```

[13]:

```
# Train classifier
print("Training classifier...")
clf = RandomForestClassifier(n_estimators=100, random_state=42,␣
 ↪class_weight='balanced')
clf.fit(X_train, y_train)
```

```
Training classifier…
```

[13]:

```
RandomForestClassifier(class_weight='balanced', random_state=42)
```

[14]:

```
# Predictions and report
y_pred = clf.predict(X_test)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99      3403
           1       1.00      0.43      0.60       173

    accuracy                           0.97      3576
   macro avg       0.99      0.72      0.80      3576
weighted avg       0.97      0.97      0.97      3576
```