# -\*- coding: utf-8 -\*-

import re

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.layers import Input, LSTM, Embedding, Dense

from tensorflow.keras.models import Model

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.preprocessing import StandardScaler

# ======================

# 数据预处理模块

# ======================

def load\_data(file\_path):

"""加载数据集示例格式：csv文件包含'url'和'label'两列"""

df = pd.read\_csv(file\_path)

return df['url'].values, df['label'].values

file\_path = "D:\\虚拟C盘\\PythonProject2\\test\_urls\_1.csv"

urls, labels = load\_data(file\_path)

print(urls, labels)

def clean\_data(urls, labels):

"""数据清洗函数"""

# 创建DataFrame以便于操作

df = pd.DataFrame({'url': urls, 'label': labels})

# 处理缺失值

df.dropna(inplace=True) # 删除包含缺失值的行

# 处理重复数据

df.drop\_duplicates(inplace=True) # 删除重复的行

# 处理异常值（例如，异常长的URL）

max\_length = 2000 # 定义正常URL的最大长度

df['url\_length'] = df['url'].apply(lambda x: len(x))

df = df[df['url\_length'] <= max\_length] # 删除过长的URL

# 统一文本格式

df['url'] = df['url'].str.lower() # 转换为小写

df['url'] = df['url'].str.strip() # 去除首尾空格

# 更新urls和labels

urls = df['url'].values

labels = df['label'].values

return urls, labels

def url\_to\_ascii(url, max\_length=200):

"""将URL转换为ASCII编码序列并填充/截断到固定长度"""

# 转换为ASCII码序列（0-127）

ascii\_seq = [ord(c) for c in url if ord(c) < 128]

# 填充/截断处理

padded\_seq = pad\_sequences([ascii\_seq], maxlen=max\_length, padding='post', truncating='post')[0]

return padded\_seq

def preprocess\_data(urls, labels, max\_length=200):

"""数据预处理主函数"""

# ASCII编码转换

X\_ascii = np.array([url\_to\_ascii(url, max\_length) for url in urls])

# 标签处理

y = np.array(labels)

return X\_ascii, y

# ======================

# LSTM模型构建（使用 Functional API）

# ======================

def build\_lstm\_model(input\_dim=128, embedding\_dim=100, lstm\_units=128):

# 显式定义输入层

inputs = Input(shape=(None,), name='input\_layer')

# Embedding 层

x = Embedding(input\_dim=input\_dim, output\_dim=embedding\_dim, mask\_zero=True)(inputs)

# LSTM 层

x = LSTM(lstm\_units, return\_sequences=False)(x)

# 特征层（命名用于后续提取）

features = Dense(64, activation='relu', name='feature\_layer')(x)

# 分类输出层

outputs = Dense(1, activation='sigmoid')(features)

# 构建完整模型

model = Model(inputs=inputs, outputs=outputs)

return model

def extract\_lstm\_features(model, X\_ascii):

"""提取LSTM高级特征"""

features = model.predict(X\_ascii)

return features

# ======================

# 主流程（修正后）

# ======================

if \_\_name\_\_ == "\_\_main\_\_":

# 1. 加载数据

urls, labels = load\_data("test\_urls\_1.csv")

# 2. 数据清洗

urls, labels = clean\_data(urls, labels)

# 3. 预处理

X\_ascii, y = preprocess\_data(urls, labels)

# 4. 划分数据集（修改后：保留URL索引）

indices = np.arange(len(urls))

train\_indices, test\_indices = train\_test\_split(

indices,

test\_size=0.2,

stratify=y,

random\_state=42

)

X\_train, X\_test = X\_ascii[train\_indices], X\_ascii[test\_indices]

y\_train, y\_test = y[train\_indices], y[test\_indices]

urls\_test = urls[test\_indices] # 获取测试集URL

# 5. 训练完整模型

lstm\_model = build\_lstm\_model(input\_dim=128, embedding\_dim=100, lstm\_units=128)

lstm\_model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

lstm\_model.fit(X\_train, y\_train, epochs=10, batch\_size=64, validation\_split=0.1)

# 6. 创建特征提取器（直接从原模型提取中间层）

feature\_extractor = Model(

inputs=lstm\_model.input, # 输入为原模型的输入

outputs=lstm\_model.get\_layer('feature\_layer').output # 输出为特征层

)

# 7. 提取LSTM特征

train\_lstm\_feats = feature\_extractor.predict(X\_train)

test\_lstm\_feats = feature\_extractor.predict(X\_test)

# 8. 标准化特征

scaler = StandardScaler()

train\_features = scaler.fit\_transform(train\_lstm\_feats)

test\_features = scaler.transform(test\_lstm\_feats)

# 9. 训练随机森林

rf = RandomForestClassifier(

n\_estimators=200,

max\_depth=15,

class\_weight='balanced',

random\_state=42

)

rf.fit(train\_features, y\_train)

# 10. 评估并计算置信度（新增部分）

y\_proba = rf.predict\_proba(test\_features) # 获取概率预测

y\_pred = rf.predict(test\_features)

confidence = y\_proba[np.arange(len(y\_pred)), y\_pred] # 动态选择对应类别的概率

# 生成结果DataFrame

results\_df = pd.DataFrame({

'url': urls\_test,

'true\_label': y\_test,

'predicted\_label': y\_pred,

'confidence': confidence

})

# 打印分类报告和置信度统计

print(classification\_report(y\_test, y\_pred, target\_names=['正常', '钓鱼']))

print("\n置信度统计：")

print(f"平均置信度: {np.mean(confidence):.2%}")

print(f"正确预测的置信度均值: {np.mean(confidence[y\_pred == y\_test]):.2%}")

print(f"错误预测的置信度均值: {np.mean(confidence[y\_pred != y\_test]):.2%}")

# 11. 保存模型和结果（新增保存预测结果）

results\_df.to\_csv("prediction\_results\_with\_confidence.csv", index=False)

feature\_extractor.save("lstm\_feature\_extractor.h5")

import joblib

joblib.dump(rf, "random\_forest\_model.pkl")

joblib.dump(scaler, "feature\_scaler.pkl")

# 12. 保存特征到CSV（原功能保留）

pd.DataFrame(train\_lstm\_feats).to\_csv("train\_lstm\_features.csv", index=False)

pd.DataFrame(test\_lstm\_feats).to\_csv("test\_lstm\_features.csv", index=False)