```
import pandas as pd
import numpy as np
import re
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder
from lime.lime_text import LimeTextExplainer
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dropout, Dense
import html

# Load and preprocess your data
df = pd.read_csv('/kaggle/input/datset8/requirment8.csv', encoding='latin1')

# Preprocessing function
def clean_text(text):
    text = BeautifulSoup(text, "lxml").get_text()  # Using BeautifulSoup to remove HTML content
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return text

df['Base_Reviews'] = df['Base_Reviews'].astype(str).apply(clean_text)

# Encode the labels (categories)
label_encoder = LabelEncoder()
df['category'] = label_encoder.fit_transform(df['category'])
class_names = label_encoder.classes_.tolist()

# Tokenizer and Pad Sequences for CNN
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(df['Base_Reviews'])
X = tokenizer.texts_to_sequences(df['Base_Reviews'])
X = pad_sequences(X, maxlen=100)
vocab_size = len(tokenizer.word_index) + 1

# Mapping categories to numerical labels
y = df['category'].values

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

# Convert labels to categorical for classification
y_train_cat = tf.keras.utils.to_categorical(y_train, num_classes=len(class_names))
y_test_cat = tf.keras.utils.to_categorical(y_test, num_classes=len(class_names))

# Define the CNN model with learning rate 0.001
def create_cnn_model():
    model = Sequential([
        Embedding(input_dim=vocab_size, output_dim=100, input_length=100),
        Conv1D(128, 5, activation='relu'),
        MaxPooling1D(pool_size=2),
        Flatten(),
```

```
        Dropout(0.5),
        Dense(len(class_names), activation='softmax')
    ])
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Implementing K-Fold Cross Validation with K=10
kfold = KFold(n_splits=10, shuffle=True, random_state=1)

for fold, (train_idx, val_idx) in enumerate(kfold.split(X_train)):
    print(f"Training fold {fold + 1}")

    # Split the training data into training and validation sets
    X_fold_train, X_fold_val = X_train[train_idx], X_train[val_idx]
    y_fold_train, y_fold_val = y_train_cat[train_idx], y_train_cat[val_idx]

    # Create and train the CNN model for each fold
    cnn_model = create_cnn_model()
    cnn_model.fit(X_fold_train, y_fold_train, epochs=10, batch_size=32, validation_data=(X_fold_val, y_fold_val),
verbose=1)

    # Optionally evaluate on validation data
    score = cnn_model.evaluate(X_fold_val, y_fold_val, verbose=0)
    print(f"Fold {fold + 1} validation accuracy: {score[1]}")

# Train the CNN model on the entire training set after cross-validation
cnn_model = create_cnn_model()
cnn_model.fit(X_train, y_train_cat, epochs=10, batch_size=32, validation_data=(X_test, y_test_cat), verbose=1)

# Initialize LIME Text Explainer
explainer = LimeTextExplainer(class_names=class_names)

# Function for CNN prediction probability
def cnn_predict_proba(texts):
    vec_texts = tokenizer.texts_to_sequences(texts)
    vec_texts = pad_sequences(vec_texts, maxlen=100)
    return cnn_model.predict(vec_texts)

# Generate LIME explanations for the first annotated review in each class from the full dataset
for class_name in class_names:
    class_id = label_encoder.transform([class_name])[0]
    class_indices = df[df['category'] == class_id].index  # Find indices of all reviews in this class

    if class_indices.size > 0:
        first_review_index = class_indices[0]  # Get the index of the first review for this class
        text_instance = df['Base_Reviews'].iloc[first_review_index]

        # Generate the LIME explanation
        exp = explainer.explain_instance(text_instance, cnn_predict_proba, num_features=10, labels=[class_id])

        # Save explanation as HTML
        html_file_name = f'Z_LIME_explanation_{class_name}.html'
        exp.save_to_file(html_file_name)
        print(f'Saved LIME explanation for class "{class_name}" to {html_file_name}')
        exp.show_in_notebook(text=True)  # Optional: Display in notebook
```

```
# Create and save the feature importance plot
num_samples = 20
aggregate_explanations = []

for i in range(num_samples):
    text_instance = df['Base_Reviews'].iloc[i]  # Get the review text
    exp = explainer.explain_instance(text_instance, cnn_predict_proba, num_features=20)
    aggregate_explanations.extend(exp.as_list())

# Calculate and plot the aggregate feature importance
feature_importances = {}
for feature, importance in aggregate_explanations:
    if feature not in feature_importances:
        feature_importances[feature] = 0
    feature_importances[feature] += importance

sorted_features = sorted(feature_importances.items(), key=lambda x: abs(x[1]), reverse=True)[:20]
features, importances = zip(*sorted_features)
colors = ['green' if x > 0 else 'red' for x in importances]
pos = np.arange(len(features)) + .5

# Plot feature importance
fig, ax = plt.subplots(figsize=(14, 10))
bars = ax.barh(pos, importances, align='center', color=colors)
ax.set_yticks(pos)
ax.set_yticklabels(features, fontsize=14)
ax.set_xlabel('Aggregate Importance', fontsize=16)
ax.set_title('Overall LIME Feature Importances (Top 20 Features)', fontsize=18)
plt.tight_layout()
plt.savefig('z_Overall_LIME_feature_importances_CNN.png', dpi=300)  # Save at 300 DPI
plt.show()
```