```
# -*- coding: utf-8 -*-
"""
Sahil Nanavaty
IST 736
Project
"""

import pandas as pd
import re

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'C:/Users/Administrator/OneDrive/Desktop/Syracuse University/Spring
2024/IST 736 (Text Mining)/Project/movies_clean 2.csv'
df = pd.read_csv(file_path)
print(df.head())

# Filter genres
genres_of_interest = ['sci-fi', 'sports', 'romance', 'crime', 'horror']
df_filtered = df[df['genre'].isin(genres_of_interest)].reset_index(drop=True)

# Display shape of filtered dataframe
print(df_filtered.shape)

# Download NLTK data for stopwords
nltk.download('stopwords')
nltk.download('wordnet')

# Preprocess text
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove numbers and punctuation
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = text.split()
    words = [word for word in words if word not in stop_words]

    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)

# Apply preprocessing
df_filtered['processed_description'] =
df_filtered['description'].apply(preprocess_text)
```

```
# Vectorize text descriptions
vectorizer = TfidfVectorizer(max_features=2000)  # TEMP: Limit features to reduce
computation time
X = vectorizer.fit_transform(df_filtered['processed_description'])

y = df_filtered['genre']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

# Define kernels and costs
kernels = ['linear', 'rbf', 'poly']
costs = [0.1, 1, 10]

# Train models
results = []
for kernel in kernels:
    for cost in costs:
        model = SVC(kernel=kernel, C=cost)
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        results.append((kernel, cost, accuracy))
        print(f'Kernel: {kernel}, Cost: {cost}, Accuracy: {accuracy}')

# Results
for result in results:
    print(f'Kernel: {result[0]}, Cost: {result[1]}, Accuracy: {result[2]}')

# Confusion Matrix
for kernel in kernels:
    for cost in costs:
        model = SVC(kernel=kernel, C=cost)
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, predictions)
        results.append((kernel, cost, accuracy))

        # Generate confusion matrix
        cm = confusion_matrix(y_test, predictions, labels=model.classes_)

        # Plot confusion matrix
        plt.figure(figsize=(10,7))
        sns.heatmap(cm, annot=True, fmt='d', xticklabels=model.classes_,
yticklabels=model.classes_)
        plt.title(f'Confusion Matrix for Kernel: {kernel}, Cost: {cost}')
        plt.xlabel('Predicted')
        plt.ylabel('True')
        plt.show()

        print(f'Kernel: {kernel}, Cost: {cost}, Accuracy: {accuracy}')
```