**MOVIE GENRE CLASSIFICATION**

university of north texas

Department of computer science

Information Retrieval and Web search

PROJECT CODE

university of north texas

Department of computer science

INFORMATION RETRIEVAL and web search

PROJECT CODE

**USING NLP TECHNIQUES**



**Movie Genre Classification**

**using Natural Language Processing Techniques**

**by**

**Group No. 14**

**Lakshmi Keerthi Gopireddy (Student ID: 11653876)**

[*LakshmiKeerthiGopireddy@my.unt.edu*](mailto:LakshmiKeerthiGopireddy@my.unt.edu)

**Major:** Artificial Intelligence

**Raghunandan Reddy Tekulapally (Student ID: 11717166)**

[*raghunandanreddytekulapally@my.unt.edu*](mailto:raghunandanreddytekulapally@my.unt.edu)

**Major:** Computer Science

**Dinesh Reddy Bollampally (Student ID: 11706437)**

[*DineshReddyBollampally@my.unt.edu*](mailto:DineshReddyBollampally@my.unt.edu)

**Major:** Computer Science

**Pavan Kumar Allam (Student ID: 11706438)**

[*pavankumarallam@my.unt.edu*](mailto:pavankumarallam@my.unt.edu)

**Major:** Computer Science

import pandas as pd

import numpy as np

import nltk

import string

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

movie\_df1 = pd.read\_csv("/content/drive/MyDrive/AAA\_IR/train\_data.txt", delimiter=":::", engine="python", names=["ID", "TITLE", "GENRE", "DESCRIPTION"])

movie\_df2 = pd.read\_csv("/content/drive/MyDrive/AAA\_IR/test\_data\_solution.txt", delimiter=":::", engine="python", names=["ID", "TITLE", "GENRE", "DESCRIPTION"])

movie\_genre\_df = pd.concat([movie\_df1, movie\_df2], ignore\_index=True)

movie\_genre\_df["YEAR"] = movie\_genre\_df["TITLE"].str.extract(r"\((\d{4})\)")

genre\_counts = movie\_genre\_df["GENRE"].value\_counts()

plt.figure(figsize=(18, 8))

genre\_counts.plot(kind="bar", color="teal")

plt.title("Genre-wise Distribution")

plt.xlabel("Genre")

plt.ylabel("Number of Movies")

plt.xticks(rotation=45)

for index, value in enumerate(genre\_counts):

    plt.text(index, value + 5, str(value), ha='center', va='bottom')

plt.show()

movies\_per\_year = movie\_genre\_df["YEAR"].value\_counts().sort\_index()

plt.figure(figsize=(20, 6))

movies\_per\_year.plot(kind="bar", color="teal")

plt.title("Number of Movies per Year")

plt.xlabel("Year")

plt.ylabel("Number of Movies")

plt.xticks(rotation=90)

plt.tight\_layout()

plt.show()

genre\_year\_counts = movie\_genre\_df.groupby(["YEAR", "GENRE"]).size().unstack(fill\_value=0)

plt.figure(figsize=(20, 4))

genre\_year\_counts.plot(kind="bar", stacked=True, colormap="tab20", figsize=(25, 8))

plt.title("Genre Distribution for Each Year")

plt.xlabel("Year")

plt.ylabel("Number of Movies")

plt.xticks(rotation=90)

plt.legend(title="Genre", bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.tight\_layout()

plt.show()

import nltk

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer, WordNetLemmatizer

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('punkt\_tab')

stemmer = PorterStemmer()

lemmatizer = WordNetLemmatizer()

def remove\_punctuation(text):

    return text.translate(str.maketrans('', '', string.punctuation))

def remove\_numbers(text):

    return ''.join([char for char in text if not char.isdigit()])

def remove\_stopwords(text):

    stop\_words = set(stopwords.words('english'))

    return [word for word in text if word.lower() not in stop\_words]

def stem\_text(text):

    return [stemmer.stem(word) for word in text]

def lemmatize\_text(text):

    return [lemmatizer.lemmatize(word) for word in text]

movie\_genre\_df['DESCRIPTION\_CLEAN'] = movie\_genre\_df['DESCRIPTION'].apply(remove\_punctuation)

movie\_genre\_df['DESCRIPTION\_CLEAN'] = movie\_genre\_df['DESCRIPTION\_CLEAN'].apply(remove\_numbers)

movie\_genre\_df['TOKENS'] = movie\_genre\_df['DESCRIPTION\_CLEAN'].apply(nltk.word\_tokenize)

movie\_genre\_df['DESCRIPTION\_NO\_STOPWORDS'] = movie\_genre\_df['TOKENS'].apply(remove\_stopwords)

movie\_genre\_df['STEMMED\_WORDS'] = movie\_genre\_df['DESCRIPTION\_NO\_STOPWORDS'].apply(stem\_text)

movie\_genre\_df['LEMMATIZED\_WORDS'] = movie\_genre\_df['DESCRIPTION\_NO\_STOPWORDS'].apply(lemmatize\_text)

movie\_genre\_df['LEMMATIZED\_DESCRIPTION'] = movie\_genre\_df['LEMMATIZED\_WORDS'].apply(lambda x: ' '.join(x))

movie\_genre\_df['TOKENS\_COUNT'] = movie\_genre\_df['TOKENS'].apply(len)

plt.figure(figsize=(12, 6))

plt.hist(movie\_genre\_df['TOKENS\_COUNT'], bins=50, color='teal', edgecolor='black')

plt.title("Distribution of Movie Descriptions Length (Number of Tokens)")

plt.xlabel("Number of Tokens")

plt.ylabel("Frequency")

plt.show()

from collections import Counter

from wordcloud import WordCloud

all\_tokens = [word for desc in movie\_genre\_df['DESCRIPTION\_NO\_STOPWORDS'] for word in desc]

word\_freq = Counter(all\_tokens)

most\_common\_words = word\_freq.most\_common(20)

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate\_from\_frequencies(word\_freq)

plt.figure(figsize=(10, 6))

plt.imshow(wordcloud, interpolation='bilinear')

plt.title("Most Frequent Words in Movie Descriptions")

plt.axis('off')

plt.show()

most\_common\_words

avg\_tokens\_per\_genre = movie\_genre\_df.groupby("GENRE")["TOKENS\_COUNT"].mean().sort\_values(ascending=False)

plt.figure(figsize=(14, 7))

avg\_tokens\_per\_genre.plot(kind='bar', color='teal')

plt.title("Average Number of Tokens per Genre")

plt.xlabel("Genre")

plt.ylabel("Average Number of Tokens")

plt.xticks(rotation=45)

plt.show()

from sklearn.preprocessing import LabelEncoder

import pandas as pd

label\_encoder = LabelEncoder()

movie\_genre\_df['GENRE\_ENCODED'] = label\_encoder.fit\_transform(movie\_genre\_df['GENRE'])

movie\_genre\_df[['GENRE', 'GENRE\_ENCODED']].head()

y = movie\_genre\_df['GENRE\_ENCODED']

import gensim.downloader as api

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

from sklearn.feature\_selection import mutual\_info\_classif

import pickle

import numpy as np

from sklearn.preprocessing import LabelEncoder

from xgboost import XGBClassifier

word2vec\_model = api.load("word2vec-google-news-300")

def get\_word2vec\_vector(text, model, vector\_size=300):

    """

    Given a text, computes the average of word vectors from Word2Vec embeddings.

    """

    words = text.split()

    word\_vectors = []

    for word in words:

        if word in model:

            word\_vectors.append(model[word])

    if word\_vectors:

        return np.mean(word\_vectors, axis=0)

    else:

        return np.zeros(vector\_size)

movie\_genre\_df['WORD2VEC\_VECTOR'] = movie\_genre\_df['LEMMATIZED\_DESCRIPTION'].apply(lambda x: get\_word2vec\_vector(x, word2vec\_model))

X\_word2vec = np.vstack(movie\_genre\_df['WORD2VEC\_VECTOR'])

y = movie\_genre\_df['GENRE\_ENCODED']

mi\_scores = mutual\_info\_classif(X\_word2vec, y)

mi\_threshold = 0.02

selected\_features = mi\_scores >= mi\_threshold

X\_word2vec\_selected = X\_word2vec[:, selected\_features]

from imblearn.over\_sampling import SMOTE

from imblearn.under\_sampling import RandomUnderSampler

from imblearn.combine import SMOTEENN

smote = SMOTE(random\_state=16)

X\_resampled, y\_resampled = smote.fit\_resample(X\_word2vec\_selected, y)

genre\_counts\_resampled = pd.Series(y\_resampled).value\_counts()

genre\_names = label\_encoder.inverse\_transform(genre\_counts\_resampled.index)

plt.figure(figsize=(18, 8))

plt.bar(genre\_names, genre\_counts\_resampled, color="teal")

plt.title("Genre-wise Distribution (Post SMOTE)")

plt.xlabel("Genre")

plt.ylabel("Number of Movies")

plt.xticks(rotation=45)

for index, value in enumerate(genre\_counts\_resampled):

    plt.text(index, value + 5, str(value), ha='center', va='bottom')

plt.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, y\_resampled, test\_size=0.2, random\_state=16)

model = XGBClassifier(objective='multi:softmax', eval\_metric='mlogloss', use\_label\_encoder=False)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred, target\_names=label\_encoder.classes\_))

with open('word2vec\_xgb\_model.pkl', 'wb') as model\_file:

    pickle.dump(model, model\_file)

with open('label\_encoder.pkl', 'wb') as le\_file:

    pickle.dump(label\_encoder, le\_file)

with open('selected\_features.pkl', 'wb') as features\_file:

    pickle.dump(selected\_features, features\_file)

with open('selected\_features.pkl', 'wb') as features\_file:

    pickle.dump(selected\_features, features\_file)

def predict\_genre(new\_data):

    new\_data\_vector = get\_word2vec\_vector(new\_data, word2vec\_model)

    new\_data\_vector = np.array(new\_data\_vector).reshape(1, -1)

    new\_data\_vector = new\_data\_vector[:, selected\_features]

    prediction = model.predict(new\_data\_vector)

    predicted\_genre = label\_encoder.inverse\_transform(prediction)

    return predicted\_genre[0]

new\_input = input("Enter a movie description: ")

predicted\_class = predict\_genre(new\_input)

print(f"Predicted Genre: {predicted\_class}")