```
!pip install implicit
```

```
Collecting implicit
  Downloading implicit-0.7.2-cp311-cp311-manylinux2014_x86_64.whl.metadata (6.1 kB)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from implicit) (2.0.2)
Requirement already satisfied: scipy>=0.16 in /usr/local/lib/python3.11/dist-packages (from implicit) (1.15.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from implicit) (4.67.1)
Requirement already satisfied: threadpoolctl in /usr/local/lib/python3.11/dist-packages (from implicit) (3.6.0)
Downloading implicit-0.7.2-cp311-cp311-manylinux2014_x86_64.whl (8.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 44.3 MB/s eta 0:00:00
Installing collected packages: implicit
Successfully installed implicit-0.7.2
```

```python
import pandas as pd
import numpy as np
import ast
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import warnings
warnings.filterwarnings('ignore')

# For implicit model
import scipy.sparse as sparse
import implicit

# Load data
movies = pd.read_csv('movie.csv')
ratings = pd.read_csv('rating.csv')
tmdb_movies = pd.read_csv('tmdb_5000_movies.csv')
tmdb_credits = pd.read_csv('tmdb_5000_credits.csv')
links = pd.read_csv('link.csv')

# Merge TMDB data
tmdb = tmdb_movies.merge(tmdb_credits, left_on='id', right_on='movie_id', how='left')
links = links.dropna(subset=['tmdbId'])
links['tmdbId'] = links['tmdbId'].astype(int)
tmdb['id'] = tmdb['id'].astype(int)
df = links.merge(tmdb, left_on='tmdbId', right_on='id')

# Functions to parse JSON-like strings
def get_director(x):
    for i in ast.literal_eval(x):
        if i['job'] == 'Director':
            return i['name']
    return ''

def get_top_cast(x):
    return [i['name'] for i in ast.literal_eval(x)[:3]]

def parse_genres(x):
    return [i['name'] for i in ast.literal_eval(x)]

# Fill missing values
df['crew'] = df['crew'].fillna('[]')
df['cast'] = df['cast'].fillna('[]')
df['genres'] = df['genres'].fillna('[]')
df['overview'] = df['overview'].fillna('')

# Extract features
df['director'] = df['crew'].apply(get_director)
df['cast'] = df['cast'].apply(get_top_cast)
df['genres'] = df['genres'].apply(parse_genres)

# Create tags column for content similarity
def create_tags(row):
    cast = ' '.join(row['cast']) if isinstance(row['cast'], list) else ''
    genres = ' '.join(row['genres']) if isinstance(row['genres'], list) else ''
    return f"{cast} {row['director']} {genres} {row['overview']}"

df['tags'] = df.apply(create_tags, axis=1)

# TF-IDF vectorizer and cosine similarity for content-based filtering
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
tfidf_matrix = vectorizer.fit_transform(df['tags'])
cosine_sim = cosine_similarity(tfidf_matrix)

# ========== IMPLICIT MODEL PART ==========

# Prepare user-item matrix for implicit feedback
# Implicit library expects (item-users) sparse matrix format (items as rows, users as columns)
# implicit library expects (item_users) sparse matrix format (items as rows, users as columns)
# So we create a sparse matrix with shape (num_items, num_users)

# Normalize ratings (optional), or just use raw ratings as confidence
ratings['implicit_rating'] = ratings['rating']  # You could do some scaling here if needed

# Create mapping for userId and movieId to indices
user_ids = ratings['userId'].unique()
movie_ids = ratings['movieId'].unique()

user_to_idx = {user_id: idx for idx, user_id in enumerate(user_ids)}
movie_to_idx = {movie_id: idx for idx, movie_id in enumerate(movie_ids)}

# Map userId and movieId to indices
ratings['user_idx'] = ratings['userId'].map(user_to_idx)
ratings['movie_idx'] = ratings['movieId'].map(movie_to_idx)

# Create sparse matrix (items x users)
# Data: ratings['implicit_rating'], Rows: movie_idx, Cols: user_idx
sparse_matrix = sparse.coo_matrix(
    (ratings['implicit_rating'].astype(float),
     (ratings['movie_idx'], ratings['user_idx']))
)

# Initialize ALS model from implicit
als_model = implicit.als.AlternatingLeastSquares(
    factors=50,  # number of latent factors
    regularization=0.1,
    iterations=20,
    random_state=42
)

# Train the ALS model
# implicit expects item-user matrix with confidence values (float)
# so we convert to csr format for efficiency
sparse_matrix_csr = sparse_matrix.tocsr()
als_model.fit(sparse_matrix_csr)

# ========== SAVE EVERYTHING ==========

# Save DataFrame
df.to_pickle('movies_df.pkl')

# Save cosine similarity matrix and vectorizer
with open('cosine_sim.pkl', 'wb') as f:
    pickle.dump(cosine_sim, f)

with open('tfidf_vectorizer.pkl', 'wb') as f:
    pickle.dump(vectorizer, f)

# Save ALS model (implicit models have their own save method)
with open('als_model.pkl', 'wb') as f:
    pickle.dump(als_model, f)

# Save mappings for inference (user/movie id to index)
with open('user_to_idx.pkl', 'wb') as f:
    pickle.dump(user_to_idx, f)

with open('movie_to_idx.pkl', 'wb') as f:
    pickle.dump(movie_to_idx, f)
```

```
100%                                    20/20 [00:53<00:00,  2.85s/it]
```

```
%%writefile app.py
import streamlit as st
import pandas as pd
import requests

# TMDb API Key
TMDB_API_KEY = '9aa7450d350e5855dbbe165c96c8f49e'  # Replace with your actual TMDb API key

def get_poster_url(title):
    """Fetch movie poster from TMDb using the movie title."""
    search_url = "https://api.themoviedb.org/3/search/movie"
    params = {
        'api_key': TMDB_API_KEY,
        'query': title
    }
    response = requests.get(search_url, params=params)
    if response.status_code == 200 and response.json()['results']:
        poster_path = response.json()['results'][0].get('poster_path')
        if poster_path:
            return f"https://image.tmdb.org/t/p/w500{poster_path}"
    return None

# Load your movie DataFrame
df = pd.read_pickle('movies_df.pkl')

st.title("IMDb-Like Movie Recommender")

# Sidebar for genre filter
st.sidebar.header("Genres")
all_genres = sorted({genre for genres_list in df['genres'] for genre in genres_list})
selected_genre = st.sidebar.selectbox("Select Genre", ["All"] + all_genres)

# Sidebar chat
st.sidebar.header("Chat with bot")
user_input = st.sidebar.text_input("Ask me for movie recommendations")

# Filter by genre
if selected_genre != "All":
    filtered_movies = df[df['genres'].apply(lambda x: selected_genre in x)]
else:
    filtered_movies = df

st.subheader(f"Movies {'in ' + selected_genre if selected_genre != 'All' else '(Random)'}")

# Show movies
if selected_genre == "All":
    display_movies = filtered_movies.sample(min(10, len(filtered_movies)))
else:
    display_movies = filtered_movies.head(10)

for idx, row in display_movies.iterrows():
    col1, col2 = st.columns([1, 4])
    with col1:
        poster_url = get_poster_url(row['original_title'])
        if poster_url:
            st.image(poster_url, width=150)
    with col2:
        st.markdown(f"### {row['original_title']}")
        st.markdown(f"**Genres:** {', '.join(row['genres'])}")
        st.write(row['overview'])
    st.markdown("---")

# Chatbot logic
if user_input:
    user_message = user_input.lower()
    st.sidebar.markdown("**Bot:**")

    if 'bored' in user_message or 'interesting' in user_message:
        recs = df.sample(5)
        st.sidebar.write("Here are some interesting movies for you:")
        for _, rec in recs.iterrows():
            st.sidebar.write(f"- **{rec['original_title']}**: {rec['overview'][:150]}...")
    else:
        # Try search by title or tags
        matched = df[df['original_title'].str.lower().str.contains(user_message) |
                     df['tags'].str.lower().str.contains(user_message)]

        # If not found, check for genres in message
        if matched.empty:
            for genre in all_genres:
                if genre.lower() in user_message:
                    matched = df[df['genres'].apply(lambda x: genre in x)]
                    break

        if matched.empty:
            st.sidebar.write("No matching movies found.")
        else:
            st.sidebar.write("Here are some movies matching your search:")
            for _, rec in matched.head(5).iterrows():
                st.sidebar.write(f"- **{rec['original_title']}**: {rec['overview'][:150]}...")
```

```
Writing app.py
```

```
import pandas as pd


df = pd.read_pickle('movies_df.pkl')
```

```
df.columns # This will print all column names on your Streamlit app
```

```
Index(['movieId', 'imdbId', 'tmdbId', 'budget', 'genres', 'homepage', 'id',
       'keywords', 'original_language', 'original_title', 'overview',
       'popularity', 'production_companies', 'production_countries',
       'release_date', 'revenue', 'runtime', 'spoken_languages', 'status',
       'tagline', 'title_x', 'vote_average', 'vote_count', 'movie_id',
       'title_y', 'cast', 'crew', 'director', 'tags'],
      dtype='object')
```