Hybrid Movie Recommendation System Using Content-Based and Collaborative Filtering with Stacking: An XGBoost Approach

We use MovieLens Latest Small Dataset - https://grouplens.org/datasets/movielens/latest/

Importing necessary libraries

```
[ ] import pandas as pd
    import numpy as np
    from sklearn.preprocessing import LabelEncoder
    from sklearn.metrics import precision_score, recall_score, f1_score
    from sklearn.model_selection import train_test_split
    import xgboost as xgb
    from sklearn.decomposition import TruncatedSVD
    from sklearn.neighbors import NearestNeighbors
    from sklearn.preprocessing import StandardScaler
    from sklearn.ensemble import RandomForestClassifier
```

Load MovieLens dataset

```
[ ] movies_df = pd.read_csv('movies.csv')
    ratings_df = pd.read_csv('ratings.csv')
```

Preprocess movie metadata for content-based filtering

```
movies_df['genres'] = movies_df['genres'].str.split('|')
genre_list = set([genre for genres in movies_df['genres'] for genre in genres])
for genre in genre_list:
    movies_df[genre] = movies_df['genres'].apply(lambda x: 1 if genre in x else 0)
```

Normalize the movie features (genres)

```
movie_features = movies_df[list(genre_list)]
scaler = StandardScaler()
movie_features_scaled = scaler.fit_transform(movie_features)
```

Collaborative filtering using SVD (Singular Value Decomposition)

```
user_movie_matrix = ratings_df.pivot(index='userId', columns='movieId', values='rating').fillna(0)
svd = TruncatedSVD(n_components=50, random_state=42)
user_movie_matrix_svd = svd.fit_transform(user_movie_matrix)
```

Fit KNN model for content-based filtering

```
knn = NearestNeighbors(n_neighbors=10, metric='cosine')
knn.fit(movie_features_scaled)
```

```
NearestNeighbors
NearestNeighbors(metric='cosine', n_neighbors=10)
```

Define LabelEncoder for user and movie IDs

```
user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()
```

Fit the label encoders on the entire ratings data

```
user_encoder.fit(ratings_df['userId'])
movie_encoder.fit(ratings_df['movieId'])
```

```
LabelEncoder
LabelEncoder()
```

Generate content-based and collaborative predictions

```
# def generate_predictions(data):
#     content_preds = []
#     collaborative_preds = []

#     for _, row in data.iterrows():
#         user_id, movie_id = row['userId'], row['movieId']

#         # Content-based prediction using KNN
#         movie_idx = movies_df[movies_df['movieId'] == movie_id].index[0]
#         distances, indices = knn.kneighbors(movie_features_scaled[movie_idx].reshape(1, -1))
#         similar_movies = indices.flatten()
#         content_based_prediction = np.mean([ratings_df[(ratings_df['movieId'] == movies_df['movieId'].iloc[i]) &
#                                                        (ratings_df['userId'] == user_id)]['rating'].mean()
#                                             for i in similar_movies])

#         # Collaborative filtering prediction using SVD
#         user_idx = user_encoder.transform([user_id])[0]
#         movie_idx_als = movie_encoder.transform([movie_id])[0]
#         collaborative_prediction = np.dot(user_movie_matrix_svd[user_idx, :], svd.components_[:, movie_idx_als])

#         content_preds.append(content_based_prediction)
#         collaborative_preds.append(collaborative_prediction)

#     return np.array(content_preds), np.array(collaborative_preds)
```

```
def generate_predictions(data):
    content_preds = []
    collaborative_preds = []

    global_avg = ratings_df['rating'].mean()

    for _, row in data.iterrows():
        user_id, movie_id = row['userId'], row['movieId']

        # Content-based prediction using KNN
        try:
            movie_idx = movies_df[movies_df['movieId'] == movie_id].index[0]
            distances, indices = knn.kneighbors(movie_features_scaled[movie_idx].reshape(1, -1))
            similar_movies = indices.flatten()

            # Try to get ratings the user gave to similar movies
            similar_ratings = [
                ratings_df[
                    (ratings_df['movieId'] == movies_df['movieId'].iloc[i]) &
                    (ratings_df['userId'] == user_id)
                ]['rating'].values
                for i in similar_movies
            ]
            similar_ratings = [r[0] for r in similar_ratings if len(r) > 0]

            # If user rated similar movies, average them; otherwise fallback
            if similar_ratings:
                content_based_prediction = np.mean(similar_ratings)
            else:
                content_based_prediction = ratings_df[ratings_df['userId'] == user_id]['rating'].mean()
                if np.isnan(content_based_prediction):
                    content_based_prediction = global_avg

        except:
            content_based_prediction = global_avg

        # Collaborative filtering prediction using SVD
        try:
            user_idx = user_encoder.transform([user_id])[0]
            movie_idx_als = movie_encoder.transform([movie_id])[0]
            collaborative_prediction = np.dot(
                user_movie_matrix_svd[user_idx, :], svd.components_[:, movie_idx_als])
        except:
            collaborative_prediction = global_avg

        content_preds.append(content_based_prediction)
        collaborative_preds.append(collaborative_prediction)

    return np.array(content_preds), np.array(collaborative_preds)
```

Prepare data for training

```
X_train, X_test = train_test_split(ratings_df, test_size=0.2, random_state=42)
content_preds_train, collaborative_preds_train = generate_predictions(X_train)
content_preds_test, collaborative_preds_test = generate_predictions(X_test)
```

```
y_true = (X_test['rating'] >= 3).astype(int)

# Content-based binary predictions
content_pred_binary = (content_preds_test >= 3).astype(int)

# Collaborative filtering binary predictions
collaborative_pred_binary = (collaborative_preds_test >= 3).astype(int)
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Content-based evaluation
precision_content = precision_score(y_true, content_pred_binary, zero_division=0)
recall_content = recall_score(y_true, content_pred_binary, zero_division=0)
f1_content = f1_score(y_true, content_pred_binary, zero_division=0)

# Collaborative filtering evaluation
precision_collab = precision_score(y_true, collaborative_pred_binary, zero_division=0)
recall_collab = recall_score(y_true, collaborative_pred_binary, zero_division=0)
f1_collab = f1_score(y_true, collaborative_pred_binary, zero_division=0)
```

```
print("Content-Based Filtering")
print(f"Precision: {precision_content:.4f}")
print(f"Recall:    {recall_content:.4f}")
print(f"F1-Score:  {f1_content:.4f}\n")

print("Collaborative Filtering (SVD)")
print(f"Precision: {precision_collab:.4f}")
print(f"Recall:    {recall_collab:.4f}")
print(f"F1-Score:  {f1_collab:.4f}")
```

```
Content-Based Filtering
Precision: 0.8921
Recall:    0.9201
F1-Score:  0.9059

Collaborative Filtering (SVD)
Precision: 0.9905
Recall:    0.3260
F1-Score:  0.4905
```

Stack content-based and collaborative predictions

```
X_stack_train = np.column_stack((content_preds_train, collaborative_preds_train))
X_stack_test = np.column_stack((content_preds_test, collaborative_preds_test))
```

Train meta-model using XGBoost (or another model)

```
meta_model = xgb.XGBClassifier(random_state=42)
meta_model.fit(X_stack_train, X_train['rating'] >= 3)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

Predict on test set

```
y_pred = meta_model.predict(X_stack_test)
```

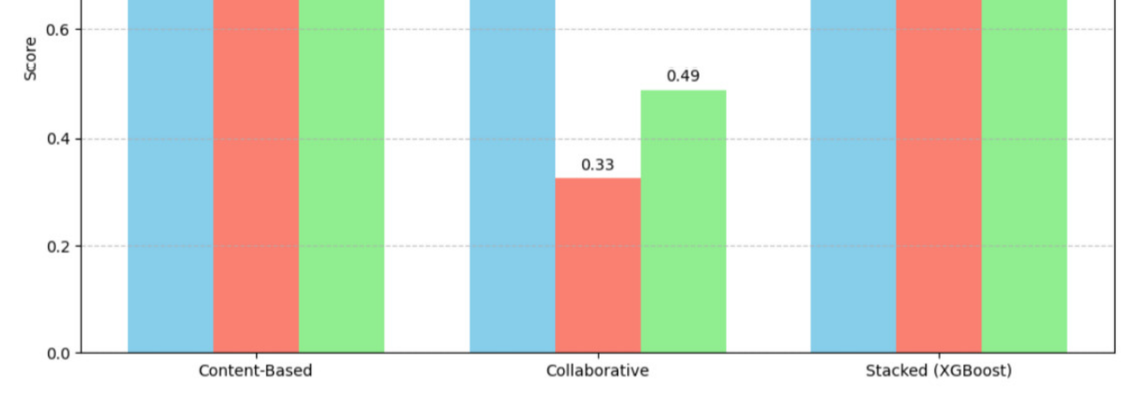
Evaluate the stacked mode

```
precision = precision_score(X_test['rating'] >= 3, y_pred)
recall = recall_score(X_test['rating'] >= 3, y_pred)
f1 = f1_score(X_test['rating'] >= 3, y_pred)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Precision: 0.8948
Recall: 0.9615
F1-Score: 0.9269
```

Overall Graph Presentation

```
import matplotlib.pyplot as plt
import numpy as np

# Data
methods = ['Content-Based', 'Collaborative', 'Stacked (XGBoost)']
precision = [0.8921, 0.9905, 0.8948]
recall = [0.9201, 0.3260, 0.9615]
f1_score = [0.9059, 0.4905, 0.9269]

x = np.arange(len(methods))
width = 0.25

# Plotting
fig, ax = plt.subplots(figsize=(10, 6))
bars1 = ax.bar(x - width, precision, width, label='Precision', color='skyblue')
bars2 = ax.bar(x, recall, width, label='Recall', color='salmon')
bars3 = ax.bar(x + width, f1_score, width, label='F1-Score', color='lightgreen')

# Labels and formatting
ax.set_ylabel('Score')
ax.set_title('Performance Comparison of Recommendation Methods')
ax.set_xticks(x)
ax.set_xticklabels(methods)
ax.set_ylim(0, 1.1)
ax.legend()
ax.grid(axis='y', linestyle='--', alpha=0.7)

# Annotate bars
for bars in [bars1, bars2, bars3]:
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')

plt.tight_layout()
plt.show()
```




```
from sklearn.metrics import mean_absolute_error

# Ground truth
y_true = X_test['rating']

# Content-based predictions
mae_content = mean_absolute_error(y_true, content_preds_test)

# Collaborative filtering predictions
mae_collab = mean_absolute_error(y_true, collaborative_preds_test)

y_pred_hybrid = meta_model.predict(X_stack_test)
mae_hybrid = mean_absolute_error((y_true >= 3).astype(int), y_pred_hybrid)

print("MAE Scores")
print(f"Content-Based Filtering MAE:     {mae_content:.4f}")
print(f"Collaborative Filtering MAE:     {mae_collab:.4f}")
print(f"Hybrid Model (XGBoost) MAE:      {mae_hybrid:.4f}")
```

```
MAE Scores
Content-Based Filtering MAE:     0.5617
Collaborative Filtering MAE:     1.5438
Hybrid Model (XGBoost) MAE:      0.1226
```

Hybrid Recommendation

Refit the label encoders on the entire dataset

```
user_encoder.fit(ratings_df['userId'])
movie_encoder.fit(ratings_df['movieId'])
```

```
LabelEncoder
LabelEncoder()
```

top_n_recommendations

```
def get_top_n_recommendations(user_id, top_n=10):
    rated_movies = ratings_df[ratings_df['userId'] == user_id]['movieId'].values

    # Generate content-based and collaborative predictions for all movies
    content_preds_all = []
    collaborative_preds_all = []

    for movie_id in movies_df['movieId']:
        # Content-based prediction using KNN
        movie_idx = movies_df[movies_df['movieId'] == movie_id].index[0]
        distances, indices = knn.kneighbors(movie_features_scaled[movie_idx].reshape(1, -1))
        similar_movies = indices.flatten()
        content_based_prediction = np.mean([ratings_df[(ratings_df['movieId'] == movies_df['movieId'].iloc[i]) &
                                                       (ratings_df['userId'] == user_id)]['rating'].mean()
                                            for i in similar_movies])

        # Collaborative filtering prediction using SVD
        try:
            user_idx = user_encoder.transform([user_id])[0]
            movie_idx_als = movie_encoder.transform([movie_id])[0]
            collaborative_prediction = np.dot(user_movie_matrix_svd[user_idx, :], svd.components_[:, movie_idx_als])
        except ValueError:
            collaborative_prediction = 0

        content_preds_all.append(content_based_prediction)
        collaborative_preds_all.append(collaborative_prediction)

    # Stack content-based and collaborative predictions
    stacked_preds = np.column_stack((content_preds_all, collaborative_preds_all))

    # Get the predictions from the meta-model
    predictions = meta_model.predict(stacked_preds)

    # Filter out movies the user has already rated
    recommendations = [(movie_id, prediction) for movie_id, prediction in zip(movies_df['movieId'], predictions)
                       if movie_id not in rated_movies]

    # Sort the recommendations by predicted relevance and get the top N
    recommendations.sort(key=lambda x: x[1], reverse=True)

    top_recommendations = recommendations[:top_n]

    # Get movie details (name, genre, id)
    movie_details = []
    for movie_id, _ in top_recommendations:
        movie_info = movies_df[movies_df['movieId'] == movie_id].iloc[0]
        movie_name = movie_info['title']
        movie_genre = ', '.join(movie_info['genres'])
        movie_details.append({
            'movieId': movie_id,
            'movieName': movie_name,
            'movieGenre': movie_genre
        })

    return movie_details
```

Test Recommendation

```
user_id = 1
top_10_recommendations = get_top_n_recommendations(user_id, top_n=10)
```

```
df_recommendations = pd.DataFrame(top_10_recommendations)

# Display the table
df_recommendations.head(10)
```

	movieId	movieName	movieGenre
0	2	Jumanji (1995)	Adventure, Children, Fantasy
1	4	Waiting to Exhale (1995)	Comedy, Drama, Romance
2	5	Father of the Bride Part II (1995)	Comedy
3	7	Sabrina (1995)	Comedy, Romance
4	8	Tom and Huck (1995)	Adventure, Children
5	9	Sudden Death (1995)	Action
6	10	GoldenEye (1995)	Action, Adventure, Thriller
7	11	American President, The (1995)	Comedy, Drama, Romance
8	12	Dracula: Dead and Loving It (1995)	Comedy, Horror
9	13	Balto (1995)	Adventure, Animation, Children

Next steps: Generate code with df_recommendations | View recommended plots | New interactive sheet

```
user_id = 10
top_10_recommendations = get_top_n_recommendations(user_id, top_n=10)

for movie in top_10_recommendations:
    print(f"Movie ID: {movie['movieId']}, Movie Name: {movie['movieName']}, Movie Genre: {movie['movieGenre']}")
```

```
Movie ID: 1, Movie Name: Toy Story (1995), Movie Genre: Adventure, Animation, Children, Comedy, Fantasy
Movie ID: 2, Movie Name: Jumanji (1995), Movie Genre: Adventure, Children, Fantasy
Movie ID: 5, Movie Name: Father of the Bride Part II (1995), Movie Genre: Comedy
Movie ID: 7, Movie Name: Sabrina (1995), Movie Genre: Comedy, Romance
Movie ID: 10, Movie Name: GoldenEye (1995), Movie Genre: Action, Adventure, Thriller
Movie ID: 11, Movie Name: American President, The (1995), Movie Genre: Comedy, Drama, Romance
Movie ID: 13, Movie Name: Balto (1995), Movie Genre: Adventure, Animation, Children
Movie ID: 17, Movie Name: Sense and Sensibility (1995), Movie Genre: Drama, Romance
Movie ID: 18, Movie Name: Four Rooms (1995), Movie Genre: Comedy
Movie ID: 26, Movie Name: Othello (1995), Movie Genre: Drama
```