business-report

December 11, 2023

1 Business Evaluation Report

To validate profitability of the recommender engine, I am simulating the business model for a specific user (I am taking user_id = 9) by calculating the expected revenue and costs based on the provided business context

- Movie Rental Fee: `$5`
- Movie Purchase Fee: `$12`
- Monthly Membership: `$20`
- Cost of Storing Uncompressed Movie: `$0.75/day`
- Cost of Movie Recommendation: `$0.01/recommended movie`
- Constraints on movie renting and purchasing: Every rented movie has a rental expiration period of 72 hours. When a rented movie is started, it must be completed in the next 24 hours. These constraints are associated with the client's expenses to store rented movies in a de-compressed format. When a purchased movie is not watched in the last 15 days, it gets compressed to reduce storage costs.

1.1 Import the required libraries

[1]:
```
import pandas as pd
import streamlit as st
from joblib import load
from datetime import datetime, timedelta
import random
```

1.2 Load files

1. Load pre-trained recommender model
2. Load ratings csv
3. Load movies csv

[2]:
```
model = load('recommendation_model.joblib')
ratings = pd.read_csv('ratings.csv')
movies = pd.read_csv('movies.csv')
```

1.2.1 Method to get recommendations

I am reusing the code I have written in the `recommend_movie.py`, it just I will be returning (movie_id, rating) instead of (movie_name, rating)

[3]:
```
def get_movie_recommendations(user_id, model, n=5):
    """
    Get Movie Recommendations for a User

    Given a user ID, a recommendation model, and an optional parameter for the␣
↪number of recommendations (default is 5),
    this function predicts and returns the top N movie recommendations for the␣
↪user.

    Parameters:
    - user_id (int): The ID of the user for whom movie recommendations are␣
↪requested.
    - model: The recommendation model used for predicting movie ratings.
    - n (int, optional): The number of movie recommendations to generate. The␣
↪default is 5.

    Returns:
    - List of tuples, each containing a movie ID and its estimated rating,␣
↪representing the top N recommended movies.
    """
    # Get a list of all movie_id
    all_movie_ids = ratings['movieId'].unique()

    # Get movie_id not rated by the user
    user_ratings = ratings[ratings['userId'] == user_id]
    user_rated_movies = user_ratings['movieId'].values
    user_unrated_movies = []
    for movie_id in all_movie_ids:
        if movie_id not in user_rated_movies:
            user_unrated_movies.append(movie_id)

    # Predict ratings for unrated movies
    predictions = [model.predict(user_id, movie_id) for movie_id in␣
↪user_unrated_movies]
    predictions.sort(key=lambda x: x.est, reverse=True)
    # Get the top N recommendations
    top_recommendations = predictions[:n]
    # top_movies_info = [(movies[movies['movieId'] == prediction.iid]['title'].
↪values[0], prediction.est) for prediction in top_recommendations]
    top_movie_ids =  [(prediction.iid, prediction.est) for prediction in␣
↪top_recommendations]
    return top_movie_ids
```

1.3 Business Evaluation

Using information provided for the business profitability analysis and declaring some constants based on that, and developed a script that can help us to keep up the profit.

Some assumptions made here, 1. The client will recommend the top 5 movies 2. User will rent 3 out of 5 movies 3. User will purchase 2 out of 5 movies

[4]:
```
RENTAL_FEE = 5
PURCHASE_FEE = 12
MONTHLY_SUBSCRIPTION_FEE = 20
COST_PER_DAY_UNCOMPRESSED_STORAGE = 0.75
COST_PER_RECOMMENDED_MOVIE = 0.01
RENTAL_EXPIRATION_PERIOD_HOURS = 72
WATCHED_THRESHOLD_DAYS = 15
```

[5]:
```
def evaluate_profitability(user_id):
    """
    Simulate User Behavior and Evaluate Profitability

    Given a user ID, this function simulates user behavior, including renting␣
↪and purchasing movies based on recommendations.
    It then calculates the total revenue and costs associated with the␣
↪simulated behavior, considering various business constraints.

    Parameters:
    - user_id (int): The ID of the user for whom behavior is simulated.

    Returns:
    - Tuple containing total revenue and total costs incurred during the␣
↪simulation.
    """
    # get recommendations
    recommended_movie_ids = get_movie_recommendations(user_id,model,n=5)
    # extract movie_ids from the results
    movie_ids = [movie_id for movie_id, _ in recommended_movie_ids]

    # randomly shuffle the movie_ids and assign the rent or make the user␣
↪purchase those movies
    random.shuffle(movie_ids)
    num_movies_rented = min(3, len(movie_ids))
    num_movies_purchased = len(movie_ids) - num_movies_rented

    # Suppose user rented 3 movies
    user_rented_movies = movie_ids[:num_movies_rented]
    # Suppose user purchased 2 movies
    user_purchased_movies = movie_ids[num_movies_rented:num_movies_rented +␣
↪num_movies_purchased]

    total_revenue = 0
    total_costs = 0
```

```
    # Calculate revenue and costs for rented movies
    for movie_id in user_rented_movies:
        rental_expiration_time = datetime.now() +␣
↪timedelta(hours=RENTAL_EXPIRATION_PERIOD_HOURS)
        # Simulate the user watching the movie within the next 24 hours
        watched_within_24_hours = datetime.now() + timedelta(hours=23)

        if watched_within_24_hours <= rental_expiration_time:
            total_revenue += RENTAL_FEE
        else:
            # Movie not watched within 24 hours, incur storage cost
            total_costs += COST_PER_DAY_UNCOMPRESSED_STORAGE *␣
↪(rental_expiration_time - watched_within_24_hours).days

    # Calculate revenue and costs for purchased movies
    for movie_id in user_purchased_movies:
        last_watched_time = datetime.now() -␣
↪timedelta(days=WATCHED_THRESHOLD_DAYS - 1)

        if last_watched_time >= (datetime.now() -␣
↪timedelta(days=WATCHED_THRESHOLD_DAYS)):
            total_revenue += PURCHASE_FEE
        else:
            # Movie not watched in the last 15 days, incur compression cost
            total_costs += COST_PER_DAY_UNCOMPRESSED_STORAGE *␣
↪WATCHED_THRESHOLD_DAYS

    # Calculate revenue from the monthly membership fee
    total_revenue += MONTHLY_SUBSCRIPTION_FEE

    # Calculate costs for recommended movies
    total_costs += COST_PER_RECOMMENDED_MOVIE * len(recommended_movie_ids)

    return total_revenue, total_costs
```

1.3.1 Validate

In here I am generating user_id randomly and then calculating total_revenue and total costs and later calculating profit, and I see with the above assumptions, profit is of $58.95 which is consistent for this recommender engine. Recommender engine is certainly making profits.

[6]:
```
# Example usage
user_id = random.randint(1, 610)
print(user_id)
revenue, costs = evaluate_profitability(user_id)

# Calculate profit
```

```
profit = revenue - costs

print(f"Total Revenue: ${revenue:.2f}")
print(f"Total Costs: ${costs:.2f}")
print(f"Profit: ${profit:.2f}")
```

```
428
Total Revenue: $59.00
Total Costs: $0.05
Profit: $58.95
```

[ ]: