CS432/532: Final Project Report

Project Title: IMDB Movie Review Analysis.

Team Member(s): Kaswankar Pushkar Sharad, Thimmaraja Vibha, Verma Isheeta

I. Problem

The IMDB Movie data set provides us with data required to analyze the factors that contribute to a movie's success. The dataset contains all the movie's information—such as its cast, director, production company, reviews, and rating. Analyzing the performance of the movie among the audience can provide insight to the filmmakers to make better movies and inspire better content creation and the producers can make informed decisions on which movie to produce. Hence by analyzing and interpreting the movie database, we will be able to identify the trends in audience behavior and preferences.

II. Software Design and Implementation

A. *Software Design and NoSQL-Database and Tools Used*

We have made use of Python as a programming language to execute NOSQL queries. We have used libraries like PyMongo that provies a python interface to Mongodb - a NOSQL database, chart.js for data visualization, HTML, CSS, JS for structuring the UI and Flask framework to interact with the backend api.

B. *Parts that you have implemented.*

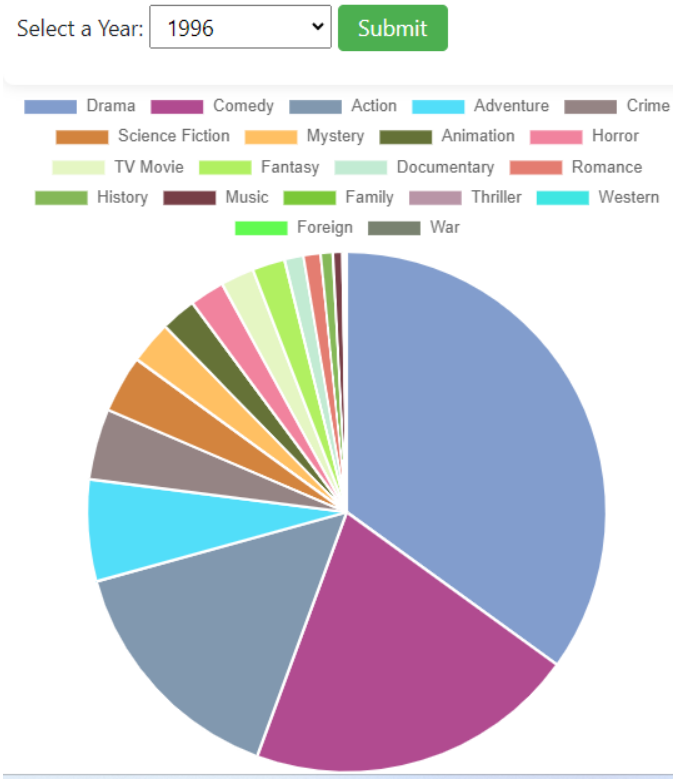
For data analysis, we have implemented pipelines using MongoDB NoSQL queries to fetch data and analyze it. Then we connected it to a Python file using PyMongo. Finally the outcome of the analysis is plotted with the help of chart.js for visual representation.

III. Project Outcome

Analysis I: To determine which genre was dominant in a particular year.

This analysis primarily focuses on determining which genre was more dominant in a particular year. Here, the user will be able to provide a year in which they want to perform the analysis. The analysis focuses on calculating the positive word count for each genre using a list of commonly used positive words on the review content column. Based on the word count, the data will be grouped together with positive word count being aggregated together and displayed in the UI with the help of a Pie Chart. In the below example, you can observe for the year 1996 ‘Drama’ was the most dominant genre which was been loved by the audience.

Below is the image of the Task-1 Analysis result.



Analysis II: Analyze the success rate of production companies by examining review content for an individual cast.

The analysis primary focuses on determining the success of a production company who has worked with an individual cast. The proposed system calculates the positive and negative word count from the review content with the help of a predefined list of positive and negative words. The results produced are grouped together based on the production company and the positive and negative word count being aggregated determines the success rate of the production company. In the below example, we can see the production company Renaissance Films got a more positive response from the audience while working with ‘Alison Goldberg’ actor. On the contrary, Cinergi Pictures Entertainment got less reviews.

Below is the image of the Task-2 Analysis result.

Analysis III: Uncovering Potential Fake Movie Reviews through Vote Average Discrepancies and Word Sentiment Analysis.

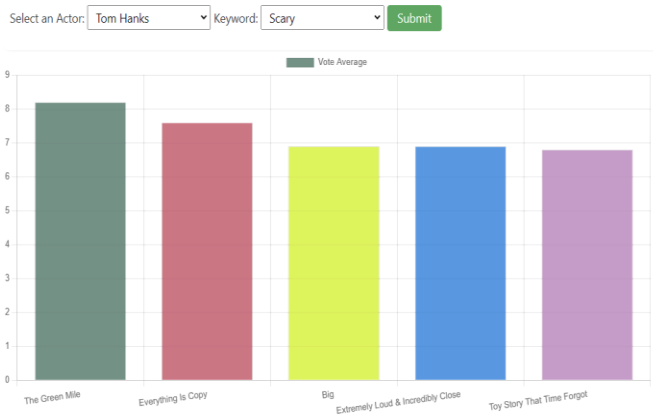
In this analysis, we can see the correlation between the vote average and review of a movie. At times a movie's review and vote average do not match, i.e., the vote count might be high, but the review would be bad and vice versa. This might happen due to multiple reasons. The primary purpose of this analysis is to help us gauge the appropriate audience review. When the user provides a movie name as an input, it analyzes the review content to calculate the positive and negative word count and based on the result it returns and displays the total count of positive and negative words along with the vote average. For example, when the movie 'The Shawshank Redemption' is given as an input, we can see the positive and negative word count by analyzing reviews of that movie and the vote average. From the given graph we can conclude that the movie reviews are legitimate as the vote average and positive words ratio matches with each other. However, for the movie 'Ravenous', we can see the negative word count is more as compared to positive word count, but the vote average is higher which clearly states that there are false reviews been added for the movie. We have projected the result with the help of a bar graph where the positive and negative word count and vote average of the respective movies are displayed to make a comparison between them.

Below are the images of the Task-3 Analysis.

Analysis IV: Sentiment Analysis to determine top 5 movies based on overview, cast, and ratings.

Through sentiment analysis, we have ascertained the top 5 movies for a given actor based on the category to which the movie belongs. The user will choose a specific actor and a keyword that describes a sentiment. We will then analyze the movie's overview using a pre-defined lists of words that are related to the keywords and will display the top 5 movies which have the overview with similar words. For instance, if we search for actor 'Tom Hanks' and the keyword "Scary," we will find the top 5 scary films starring Tom Hanks. A bar graph is used to graphically depict the outcome, with the title of the film and its vote average shown in accordance.

Below is the image of Task Analysis IV



Source Code:

For Analysis I:

positive_words_list = ["awesome", "awe-inspiring", "authentic", "beautifully crafted", "best", "bold",
"brilliance",
"brilliant", "captivating", "charming", "charming", "cinematic excellence", "clever", "cleverness", "compelling", "courageous", "creatively", "delight", "delightful", "dazzling", "discerning", "dynamic", "emotional depth", "energetic",

```
"engaging", "entertaining", "enthralling", "enchanting", "enjoyable", "enthralling", "exceptional", "exceptional", "excitement", "exciting", "excellent", "exhilarating", "expertly executed", "extraordinary", "fantastic", "flawless", "fulfilling", "fun", "good", "great", "gripping", "heartwarming", "hilarious", "humorous", "impactful", "impressive", "ingenious", "ingenious", "insightful", "inspiring", "inspirational", "intuitive", "invigorating", "joyful", "llyrical", "magical", "masterpiece", "memorable", "mesmerizing", "minds", "monumental", "must-watch", "nostalgic", "nuanced", "optimistic", "original", "outstanding", "outstanding", "peerless", "perfect", "performance", "performances", "phenomenal", "poignant", "popular", "powerful", "profound", "progressive", "radiant", "refreshing", "relevant", "remarkable", "resonant", "revolutionary", "resonant", "riveting", "satisfying", "seamless storytelling", "sincere", "stellar", "stellar performances", "strange", "strange powers", "stunning", "success", "successful", "superb", "suspenseful", "sweet", "talented", "thought-provoking", "thoughtful", "thrilling", "timely", "touching", "transcendent", "unforgettable", "unimaginative", "unparalleled", "unrivaled", "unsurpassed", "uplifting", "visionary", "visuals", "visually stunning", "witty"]

negative_words_list = ["abominable", "absurd", "absurd", "amateurish", "annoying", "appalling", "appalling",
            "atrocious", "awkward", "bad", "bland", "boring", "cheesy", "childish", "clichéd", "clumsy", "confusing", "contrived", "contrived", "corny", "cringe-worthy", "cynical", "deplorable", "despondent", "detestable", "diabolical", "disappointing", "disgusting", "dismal", "dreadful", "dull", "dumb", "execrable", "forgettable", "forced", "ghastly", "grim", "gross", "grating", "hackneyed", "haphazard", "horrendous", "irksome", "irritating", "juvenile", "laughable", "ludicrous", "lackluster", "lifeless", "miserable", "messy", "melancholy", "monstrous", "obnoxious", "offensive", "offensive", "off-putting", "overrated", "pointless", "ponderous", "poorly executed", "predictable", "preposterous", "pretentious", "repellent", "repugnant", "repulsive", "repulsive", "ridiculous", "silly", "sloppy", "sordid", "stale", "stupid", "subpar", "sucky", "sucks", "suspenseful", "tacky", "tedious", "terrible", "thoughtless", "tragic", "trite", "unappealing", "unconvincing", "underwhelming", "unimaginative", "uninspiring", "unpleasant", "unrealistic", "unrewarding", "unsatisfying", "unsophisticated", "unwatchable", "weak plot", "worse", "worst", "worst", "wretched"]

For Analysis I:

@app.route('/submit-task-1', methods=['POST'])
def get_genre_result():
    selected_year = request.form.get('selected_year')

pipeline = [
        {"$match": {"release_date": {"$regex": selected_year}, "genres": {"$regex": ""}}},
        {"$project": {"genres": 1, "review_content": 1}},
        {"$addFields": {"positive_words_count": {"$size": {"$filter": {"input": {"$split": [{"$toString": "$review_content"}, " "]}, "as": "word", "cond": {"$in": ["$$word", positive_words_list]}}}}}},
        {"$group": {"_id": "$genres", "positive_words_count": {"$sum": "$positive_words_count"}}},
        {"$addFields": {"genre_name": {"$ifNull": ["$_id", "Unknown"]}}},
        {"$project": {"_id": 0, "genre_name": 1, "positive_words_count": 1}},
        {"$sort": {"positive_words_count": -1}}
    ]
    # Execute the aggregation pipeline
    result = list(collection.aggregate(pipeline))


    # Process the selected year (you can replace this with your own logic)
    response_data = {'result': result, 'message': 'Task 1 data sent successfully'}

    # Return the response in JSON format
    return jsonify(response_data)

For Analysis II:

@app.route('/submit-task-2', methods=['POST'])
def get_actor_data():
    selected_actor = request.form.get('selected_actor')

    pipeline = [
        {"$match": {"cast": {"$regex": selected_actor}}}, # Filter by year
        {"$project": {"production_companies": 1, "review_content": 1, "budget_musd":1}},  # Project only necessary fields
        {"$addFields": {"positive_words_count": {"$size": {"$filter": {"input": {"$split": [{"$toString": "$review_content"}, " "]}, "as": "word", "cond": {"$in": ["$$word", positive_words_list]}}}}}},  # Count positive words
        {"$addFields": {"negative_words_count": {"$size": {"$filter": {"input": {"$split": [{"$toString": "$review_content"}, " "]}, "as": "word", "cond": {"$in": ["$$word", negative_words_list]}}}}}},  # Count positive words
        {"$group": {"_id": "$production_companies", "positive_words_count": {"$sum": "$positive_words_count"}, "negative_word_count":{"$sum": "$negative_words_count"}, "total_budget":{"$sum":"$budget_musd"}}},  # Group by genre and sum positive words
        {"$addFields": {"production_company": "$_id"}}, # Add a field for genre_name
        {"$project": {"_id": 0, "production_company": 1, "positive_words_count": 1, "negative_word_count":1, "total_budget":1}},  # Project the final fields
```

```
        {"$sort": {"positive_words_count": -1}}  # Sort by positive_words_count in descending order
    ]

    # Execute the aggregation pipeline
    result = list(collection.aggregate(pipeline))

    # Process the selected year (you can replace this with your own logic)
    response_data = {'result': result, 'message': 'Task 2 data sent successfully'}

    # Return the response in JSON format
    return jsonify(response_data)
```

For Analysis III:

```
@app.route('/submit-task-3', methods=['POST'])
def get_movie_data():
    selected_movie = request.form.get('selected_movie')

    pipeline = [
        {"$match": {"movie_title": {"$regex": f"^{selected_movie}$", "$options": "i"}}},  # Exact match on movie_name
        {"$project": {"vote_average": 1, "movie_title": 1, "review_content": 1}},  # Project only necessary fields
        {"$addFields": {"positive_words_count": {"$size": {"$filter": {"input": {"$split": [{"$toString": "$review_content"}, " "]}, "as": "word", "cond": {"$in": ["$$word", positive_words_list]}}}}}},  # Count positive words
        {"$addFields": {"negative_words_count": {"$size": {"$filter": {"input": {"$split": [{"$toString": "$review_content"}, " "]}, "as": "word", "cond": {"$in": ["$$word", negative_words_list]}}}}}},  # Count positive words
        {"$group": {"_id": "$movie_title", "positive_words_count": {"$sum": "$positive_words_count"}, "negative_words_count":{"$sum": "$negative_words_count"}, "vote_average":{"$first": "$vote_average"}}},  # Group by genre and sum positive words
        {"$addFields": {"movie_title": "$_id"}},  # Add a field for genre_name
        {"$project": {"_id": 0, "movie_title": 1, "positive_words_count": 1, "negative_words_count":1, "vote_average":1}},  # Project the final fields
        {"$sort": {"positive_words_count": -1}}  # Sort by positive_words_count in descending order
    ]

    # Execute the aggregation pipeline
    result = list(collection.aggregate(pipeline))

    # Process the selected year (you can replace this with your own logic)
    response_data = {'result': result, 'message': 'Task 3 data sent successfully'}

    # Return the response in JSON format
    return jsonify(response_data)
```

For Analysis IV:

```
@app.route('/submit-task-4', methods=['POST'])
def get_movie_with_keyword():
    selected_actor = request.form.get('selected_actor')
    selected_keyword = request.form.get('selected_keyword')
    print(selected_actor)
    print(selected_keyword)

    # Define the lists for each genre
    keyword_lists = {
        "kill": ["kill", "crime", "criminal", "hunted", "murder", "avenge", "attackers", "assaulted", "spy",
                 "fight", "money", "accused", "prison",
                "mysterious", "death", "police", "agent", "mission", "mystery", "detective", "victim", "steals", "revenge", "cop", "rape",
                "gangster", "drug", "plan", "enemy", "rob", "violence", "mafia", "action", "violent", "abused", "crime", "thief", "desperate"],

        "love": ["love", "romantic", "romance", "relationship", "girlfriend", "boyfriend", "happy", "loves",
                 "loved", "interested", "heart",
                "crush", "like", "friendship", "dumped"],

        "horror": ["vampire", "supernatural", "dark", "hunger", "zombies", "danger", "horrific", "murders",
                   "suspicious", "corpses",
                "monster", "death", "bloody", "soul", "rebirth", "deceased", "suicide", "frightening", "sinister", "suspect", "killer",
                "cop", "killed", "hunt", "hunter"],

        "documentary": ["history", "community", "territory", "fighter", "warrior", "mankind", "victory",
                        "war", "historic", "rebels", "government",
                "historical", "believers", "honorary", "brave", "national", "festival" ,"force", "struggle", "ancestral", "dignity","legacy",
                "legendary", "power", "century", "politics", "culture","mission", "mockumentary"],

        "thrill": ["adventure", "thrill", "magical", "creatures", "team", "discover", "agent", "mystical",
                   "heroic", "fate", "boxer", "weapon",
                "survivor", "journey", "storm", "mission", "shooting", "living", "remote", "tale", "mountaineer", "mob", "age",
```

```
            "opportunity", "relocate", "fortune", "misadventure"
,"revenge"]
    }

    # Get user input for the desired genre
    selected_list =  []
    if selected_keyword not in keyword_lists.keys():
        print("no genre in dict")
    else:
        selected_list = keyword_lists[selected_keyword]

    # MongoDB aggregation pipeline
    pipeline = [
        {
            "$match": {
                "cast": {"$regex": selected_actor, "$options": "i"},
                "overview": {
                    "$regex": "|".join(selected_list),
                    "$options": "i"
                }
            }
        },
        {
            "$group": {
                "_id": "$movie_title",
                "vote_average": {"$first": "$vote_average"}
            }
        },
        {
            "$project": {
                "movie_title": "$_id",
                "vote_average": 1,
                "_id": 0
            }
        },
        {
            "$sort": {"vote_average": -1}
        },
        {
            "$limit": 5
        }
    ]

    # Execute the query
    result = list(collection.aggregate(pipeline))

    # Process the selected year (you can replace this with your
own logic)
    response_data = {'result': result, 'message': 'Task 4 data sent
successfully'}

    # Return the response in JSON format
    return jsonify(response_data)
```

REFERENCES

1. https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/.
2. https://www.geeksforgeeks.org/mongodb-regex/
3. https://www.chartjs.org/
4. https://www.kaggle.com/code/atulpadalia/imdb-dataset-analysis/input?select=movies_complete.csv
5. https://www.digitalocean.com/community/tutorials/how-to-use-mongodb-in-a-flask-application
6. https://www.statology.org/mongodb-split-string/
7. https://www.tutorialsteacher.com/mongodb/aggregation