CMPE343 Tune Duel Project Report

Team Rocket

Yusuf Tamer Akyol, Üstün Yılmaz, Enes Hamza Üstün

December 2025

1 Project Description

In this project, we are given a dataset of songs from the music streaming app Spotify, which includes their certain attributes like artist names and danceability values. We are also given user ratings corresponding to some songs ranging from 1 star to 5 stars based on their likings.

We are asked to parse the dataset and compute certain probabilities, like how likely a song is to receive 5 stars based on its attributes. We are then asked to recommend songs to hypothetical users to find their time-until-5-stars (T_u) values and test certain hypotheses about this value and its functions (mean, variance, etc.). After that, we are asked to develop two different recommendation models that can recommend songs from the dataset to a user based on their ratings of certain songs. Finally, these recommendation models will be compared using a Monte Carlo experiment to evaluate their comparative performances based on certain metrics.

2 Part 1: Conditional Probability Modeling - Üstün

2.1 Description

In this part of the project; we are asked to estimate how likely a song in `tracks.csv` file is rated 5 stars by users in `ratings.csv` file, depending on its features (how happy, aggressive, explicit, etc. it is). We are asked to do this in **three different forms**, and then generalize it to analyze group and member preferences along with global ones. The three different forms of analysis and how they are achieved are summarized in the **Methods** section, below.

2.2 Methods

First, we are asked to estimate conditional probabilities of a song receiving 5 stars given a specific feature of it. To reach this goal, we apply Laplace Smoothing using the following line of code in our `compute_p5_given_feature` method:

```
probs["prob5 | f"] = (probs["total_5count"] + laplace) / (probs["total_count"] +
                                    laplace * 2)
```

Here, `probs` is a pandas dataframe, and utilizing the easy-to-use math operations provided by the pandas library, we can easily apply Laplace Smoothing and normalize our probabilities according to the formula:

$$P(y \mid x) = \frac{N_{y,x}+\alpha}{N_x+K\alpha}$$

Secondly, we are asked to estimate the same conditional probability but for combinations of features, which are easily handled using Python lists in the same `compute_p5_given_feature` method.

Third, we are asked to flip the estimated probability (i.e. estimate how likely a song has certain features given that it is rated 5 stars) using Naive Bayes' Rule. We define a new method `compute_feature_given_p5` that works the exact same as the prior, but with Bayes' rule implemented in this snippet of code:

```
probs["f | prob5"] = probs["prob5 | f"] * probs["probfeature"] / probs["prob5"]
                        sums = probs["f | prob5"].sum()
                                if sums > 0:
                            probs["f | prob5"] /= sums
```

Here, we use the Bayes rule formula given below and later normalize the results (All dataframes and probabilities have been previously smoothed using Laplace. No need for further smoothing.):

$$\tilde{P}(X = x \mid Y = y) = \frac{P(X=x|Y=y)}{\sum_x P(X=x|Y=y)}$$

Last but not least, we are asked to estimate analogous probabilities of group members and overall group session. This is only done for `compute_p5_given_feature` and not the other way around. In the method `personal_and_group_analysis`, which accepts a list of dataframes for each user computed using `compute_p5_given_feature`; we merge the given dataframes and average out using the following mean formula to get the group probability:

$$P_{\text{group}}(Y = 5 \mid X = x) = \frac{1}{M} \sum_{m=1}^{M} P_m(Y = 5 \mid X = x)$$

2.3 Results and Interpretations

- We can see that (for the given global `tracks.csv`), highest P(5* | primary_artist_name) are as follows:
  - Two Door Cinema Club: 0.888889
  - Milky Chance: 0.888889
  - Franz Ferdinand: 0.818182

  However, these results do not tell us much since these artists likely have low number of songs in the tracks file. We have to look at P(primary_artist_name | 5*) to see which artists have got the most 5 star ratings:
  - Taylor Swift: 0.029399
  - Adele: 0.019463
  - Ariana Grande: 0.014087
- We can also see that more popular tracks have been voted as 5 stars (P(track_popularity | 5*)) by our group, the top three being:
  - 82: 0.583333
  - 81: 0.505556
  - 87: 0.500000

The global ratings also agree with that, the top three being:

  - 80: 0.099521
  - 83: 0.076823
  - 78: 0.071924

- We can also see that `explicit` feature is the most detrimental in people rating a song 5 stars in the global data set, given that:

$$P(5 * |explicit) = 0.045 \wedge P(5 * |\neg explicit) = 0.955$$

- Last but not least, even though we are working with a very small dataset when analyzing personal and group sessions, we can see that they are consistent within but differ drastically with the global dataset when analyzing certain features, let's take `explicit` for this one because it is easy to analyze:

```
Personal and Group Analysis of P(5* | explicit):
Pm(prob5 | f)M_1 Pm(prob5 | f)M_2 Pm(prob5 | f)M_3 Pgroup(prob5 | f)
            True 0.428571 0.500000 0.200000 0.376190
           False 0.285714 0.384615 0.285714 0.318681
```

  Whereas for the global data we have $P(5 * |explicit) = 0.045 \wedge P(5 * |\neg explicit) = 0.955$ as mentioned above. Indicating that the group preferences favor explicit songs by a significant margin when compared to global data.

3 Part 2: User Variability Modeling - Yusuf

3.1 Introduction

In this section, we analyze user patience and pickiness for songs. We explore this by defining p as the probability of someone giving a song 5 star review. To find that we use 2 different methods. First we assume p is constant for everyone. Second we assume p is variable for everyone and has a Beta distribution. We define a random variable T_u, that is first time a person gives 5 star to a song. From this random variable we calculate the value of p or the α, β values for the distribution. Then we analyze the results. Additionally, we perform a hypothesis testing between 2 distinct groups of A: people who like popular songs and B: people who like niche songs. And we see if their patience differ.

3.2 Implementation

We implemented this part using pandas, numpy, math and matplotlib.plot libraries in Python. The methods and what they do is as follows:

- `get_time_to_favorite(ratings_df)`: Gets ratings path as input and calculates T_u for each user excluding the ones who don't have a 5 star rating. And return a DataFrame that has columns "user_id" and "Tu".

- `fit_geometric(tu_series)`: Calculates the p value by getting T_u series for geometric distribution using mean value and the formula $1/T_u$.

- `geometricDist(ratings_df)`: Gets ratings and returns p value and T_u DataFrame for geometric distribution.
- `geometricPMF(x, p_geo)`: Behaves as geometric mass function.
- `lbeta_manual(a, b)`: Computes log(Beta(a, b)) using log-gamma function.
- `beta_geometric_neg_log_lik(params, tu_data)`: Computes Negative Log Likelihood for Beta-Geometric Distribution for a given α, β and T_u DataFrame.
- `coordinate_descent(func, data, ...)`: Numerical optimization algorithm for finding the α, β values. Gets the cost function, and the T_u DataFrame. Optionally, starting point, step size, tolarence rate, max iterations.
- `bgDist(func, tu_values)`: With given optimization function and T_u DataFrame calculates α, β values.
- `model(ratings_df)`:Calculates Geometric and Beta-Geometric Distribution and plots it.
- `mann_whitney_manual(x, y)`: Gets 2 different user groups in the format of T_u DataFrame and applies Mann-Whitney U test (two-sided). Returns: U statistic, p-value.
- `hypothesisTesting(ratings, tracks)`: Creates 2 distinct groups of A: people who like popular songs and B: people who like niche songs. Then calls mann_whitney_manual(x, y) for them prints the results.
- `main()`: Loads ratings.csv and tracks.csv files. Calls model() and hypothesisTesting() functions and finishes the program.

3.2.1 Geometric Model (Homogeneous Users)

The Geometric model assumes that every user shares the same constant probability p of finding a favorite song in any given round. Under this assumption, T_u follows the distribution:

$$P(T_u = t) = (1-p)^{t-1}p$$

Using the MLE, where $\hat{p} = 1/\bar{T_u}$, we obtained the following results from our dataset of 291 valid users:

- **Average rounds to favorite ($\bar{T_u}$):** 3.69
- **Estimated Parameter ($\hat{p}$):** 0.2707

Fit Analysis: As shown in Figure 1 (red dashed line), the Geometric model captures the general decaying trend of the data. However, it noticeably underfits the peak at $T_u = 1$. The observed data shows that over 35% of users find a favorite in the very first round, whereas the Geometric model predicts only roughly 27%. This suggests that assuming a single constant p for all users fails to capture the behavior of "easy-to-please" users who find favorites almost immediately.

3.2.2 Beta-Geometric Model (Heterogeneous Users)

To address the limitations of the Geometric model, we applied the Beta-Geometric model. This model, unlike Geometric model, assumes each user has different patience and different success probability p_u drawn from Beta distribution with parameters α, β.

Derivation of the Beta-Geometric PMF: To derive the marginal distribution of T_u, we integrate the Geometric likelihood over the Beta prior for p:

$$P(T_u = t) = \int_0^1 P(T_u = t|p) \cdot f(p|\alpha, \beta)\, dp$$

Substituting the Geometric PMF $(1-p)^{t-1}p$ and the Beta PDF $\frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(\alpha,\beta)}$:

$$P(T_u = t) = \int_0^1 (1-p)^{t-1}p \cdot \frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(\alpha,\beta)}\, dp$$

Rearranging terms:

$$P(T_u = t) = \frac{1}{B(\alpha,\beta)} \int_0^1 p^{(\alpha+1)-1}(1-p)^{(\beta+t-1)-1}\, dp$$

The integral represents the Beta function $B(\alpha+1, \beta+t-1)$. Thus, the closed-form PMF is:

$$P(T_u = t) = \frac{B(\alpha+1, \beta+t-1)}{B(\alpha,\beta)}$$

Numerical Optimization: Since there is no simple closed-form MLE for α and β, we used our `coordinate_descent` function to minimize the negative log-likelihood defined in `beta_geometric_neg_log_lik`. We estimated:

- α: 3.0530
- β: 5.6068

Fit Analysis: The Beta-Geometric model (Figure 1, blue solid line) provides a significantly better fit to the empirical data than the simple Geometric model.

- **Peak Fit:** It accurately matches the high proportion of users finding a hit at $T_u = 1$ (approx. 0.35 probability).
- **Tail Fit:** It accounts for the "long tail" of picky users who require many rounds to find a favorite, which the simple Geometric model tends to underestimate.

The estimated parameters suggest a mean success probability of $E[p] = \frac{\alpha}{\alpha+\beta} \approx 0.35$. This is higher than the Geometric $\hat{p} \approx 0.27$, indicating that while the "average" user is quite likely to find a hit quickly, the high variance in user behavior (captured by the Beta distribution) drags the overall average waiting time ($\bar{T_u}$) up.

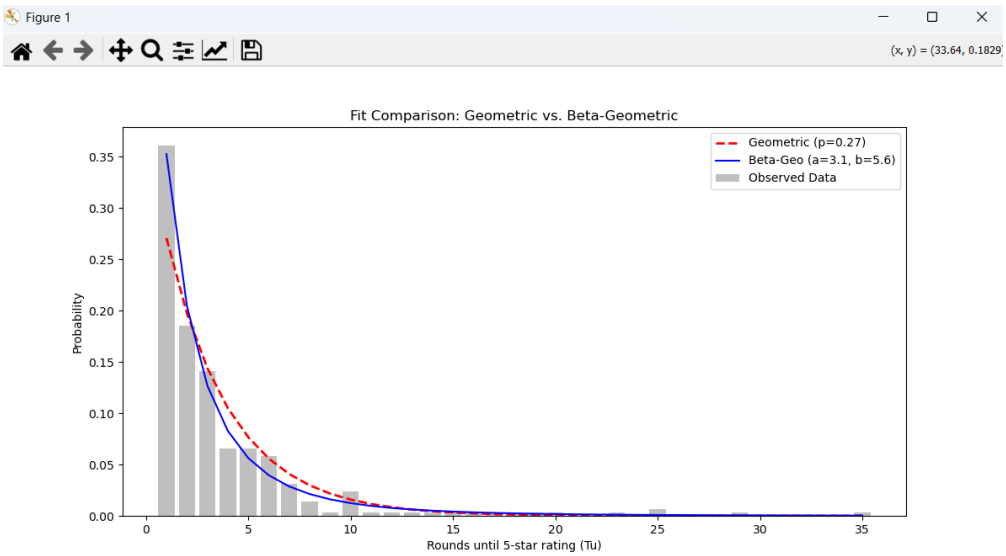

Figure 1: Fit Comparison: Geometric vs. Beta-Geometric. The Beta-Geometric model (blue) captures the initial peak and long tail better than the Geometric model (red).

3.3 Hypothesis Testing: Popular vs. Niche Tastes

We investigated whether users who prefer "Popular" tracks find their favorites faster than those who prefer "Niche" tracks.

Group Selection Algorithm: To divide users we first identified their "favorite song" as the first 5 star rated song for each user. Then from the tracks.csv we calculated the the median of the popularity of the songs and we placed each person to either group A or group B according to:

- **Group A (Popular Tastes):** Users whose favorite song had a popularity score $\geq$ the median ($n = 156$).
- **Group B (Niche Tastes):** Users whose favorite song had a popularity score $<$ the median ($n = 135$).

Hypothesis Formulation:

- $H_0 : \mu_A = \mu_B$ (There is no difference in the mean time-to-favorite between groups).
- $H_1 : \mu_A \neq \mu_B$ (There is a significant difference).

Methodology: Since the distribution of T_u is right skewed and non-normal we used Mann-Whitney U test for the Hypothesis Testing. Our implementation, `mann_whitney_manual`, ranks all observations from both groups together, computes the rank sum for one group, and derives the U statistic and p score to assess whether one population differs from one another. The p value derived using the normal distribution since we have a large number of people. **Results:**

- **Mean T_u (Group A):** 3.69
- **Mean T_u (Group B):** 3.70
- **Mann-Whitney U Statistic:** 10015.0

- **p-value:** 0.47189

Conclusion: With a p-value of 0.47 (well above the significance level of 0.05), we fail to reject the null hypothesis. The results indicate that there is no statistically significant difference in "patience" or "time-to-favorite" between users who prefer popular music and those who prefer niche music. Both groups take, on average, practically the same number of rounds (≈ 3.7) to encounter a song they love.

3.4 Discussion

The analysis confirms that user heterogeneity is a critical factor in our recommendation system. The superiority of the Beta-Geometric model proves that users are not uniform; some are "easy wins" (high p_u) while others are highly selective (low p_u).

4 Part 3: Recommender Design - Enes & Üstün

4.1 Description

In this part of the project, we are asked to implement **two types of recommenders** based on user ratings. These recommenders will take a series of user-rated songs as input, and will output several song recommendations based on that user's likings. The first type of recommender will be implemented as a **deterministic, conditional filtering** design, while the second will be implemented as a **probabilistic, utility based** design.

4.2 Conditional Filtering Recommender

This recommender evaluates each candidate track by conditioning it based on the user's past ratings, and estimating a utility value by comparing its features to the ones of the user's liking. It then orders the tracks based on their final utility scores and returns a sorted pandas dataframe. The workflow can be described as follows:

1. **Estimating user patience:** We use the `get_user_patience` method from Part 2 to interpret the user's patience. If they have low patience (T_u), we recommend more popular content to them. Vice versa, they are open to more niche, exploratory content.

2. **Feature importance calculation:** We estimate multinomial probability distributions of ratings conditioned on feature values to assign an importance to each feature which will later affect utility.

3. **Computing net utility for feature:** We then produce a table of expected utilities of features using the user's ratings. We use Laplace Smoothing for fairer calculation.

4. **Track utility scoring:** Using the feature-based utilities we calculated using the user's data and patience, we aggregate the unrated track's score by summing up each feature value multiplied by that feature's utility. The data is then ready to be sorted and displayed.

The pros and cons of this recommender can be listed as follows:

- **Pros:**
  - Strong personalization of recommendations

  - Stable and deterministic song rankings
  - Good for users that generally don't like randomness and exploration
  - Easy to implement and fits the common case (people don't really like listening to niche songs, hence why they're niche)

- **Cons:**
  - When the tastes of a user changes, it misses to capture that change
  - No exploration and randomness whatsoever, may bore some users who like listening to new and niche songs
  - Cold-start user ratings may get wrong/weak recommendations

4.3 Utility Based Probabilistic Recommender

This recommender estimates the probability of a song being liked by the user and builds a probability distribution upon that estimation. It then samples from this distribution and outputs sorted recommendations based on the sampled songs and a controlled factor of exploration with randomness. The workflow can be described as follows:

1. **Feature importance calculation:** How a feature helps predict if a song is rated 5 stars is calculated using the `calculate_importance` method.
2. **User-specific conditional probabilities and utility estimation:** For each feature, P(5* | f, user) is calculated. This value is then used to estimate the utility of an unrated track similarly to the previous recommender. However, no normalization/smoothing is done as of now, as these are not yet final calculated probabilities.
3. **Amplification of utility:** The higher utilities are scaled so that the top recommendations stand out more compared to lower ranked ones, reducing randomness.
4. **Probability normalization:** The utilities are then normalized to probabilities, so that we can use them for our recommendations.
5. **Patience based exploration:** Just like the previous recommender, we then calculate the T_u value of the user, but for a different purpose. If the user is more patient, we introduce more randomness to our recommendations. Vice versa, we reduce the randomness.
6. **Random sampling and recommendation:** We then randomly sample from the recommendation set without replacement and build the final dataframe.

The pros and cons of this recommender can be listed as follows:

- **Pros:**
  - Encourages discovery and exploration of new tastes
  - Avoids recommendation stagnation
  - Fits the behavior of a more patient, explorative audience

- **Cons:**
  - Not deterministic nor stable
  - Worse recommendations for low number of recommendations (like top-3)
  - Not suitable for impatient users that don't like randomness or exploration

5 Part 4: Monte Carlo Evaluation

5.1 Introduction

In this section, we evaluate the performance of our two recommendation models **Conditional Filtering** and **Utility-Based Sampling** by using Monte Carlo experiment. By simulating thousands of recommendation scenarios, we try to get statistically good performance estimates.
The evaluation compares the models according to three key performance metrics:

- **Hit@k**: Proportion of users who achieve at least one 5 star hit within the first k recommendations. It indicates how efficiently each model surfaces songs that users love.
- **Average Rating**: The mean rating assigned to the recommended songs, averaged over all users and recommendation rounds. Higher average ratings suggest higher overall satisfaction and recommendation quality.
- **Time-to-5⋆**: Number of recommendations needed to find a 5-star song for a user.

By comparing these metrics between the two models using confidence intervals, we try to identify which model is better at recommending songs that fit the user's taste.

5.2 Implementation

We conducted the Monte Carlo experiment by replaying the existing synthetic sessions in *ratings.csv*. The functions implemented for this process are listed below:

- **train_and_test_split(merged, test_ratio=0.2)**: Splits the data into train and test data. Merged is the input DataFrame, and test_ratio is the proportion used for splitting.
- **compute_hit_at_k(recommendations_df, test_songs_df, k)**: Computes whether at least one test song appears in top-k recommendations.
- **compute_average_rating(recommendations_df, test_songs_df, k)**: Computes average rating of recommended songs that are in the test data.
- **compute_time_to_five_star(recommendations_df, test_songs_df, max_search=100)**: Finds the position of the first 5-star song in recommendations. It searchs the 5 starred song in the first 100 recommendations (default) as the calculation of more recommendations gets infeasible.
- **monte_carlo_simulation(merged, tracks, ratings, n_simulations=1000, k=5, max_search=100)**: Runs Monte Carlo simulation to evaluate both recommendation models.
- **compute_confidence_interval(data, confidence=0.95)**: Computes mean and 95% confidence interval (default) for a list of values.
- **compare_models(results_model_a, results_model_b, metric_name)**: Compares two models on a specific metric by computing means and differences.

5.2.1 Data Splitting

In this experiment, the `train_and_test_split` function chooses a random user from the dataset for each round. We split this user's ratings into two parts based on a given test ratio (0.2 for default): 80% of the data is used to train the models, and the remaining 20% is kept hidden as a test set. This test set is used to predict whether the recommended songs fit the user's actual preferences. For each round, we check whether the recommended songs include a subset of the test set.

5.2.2 Simulation Logic

The simulation executes a loop for a given number of rounds to ensure statistical significance. In each iteration:

1. A random user is chosen, and their data is split into training and test data.
2. Both *Conditional* and *Utility* models generate a list of recommendations based on the training data.
3. The system calculates the performance metrics (Hit Rate, Rating, Time-to-5⋆) by comparing these recommendations with the test set.
4. Finally, we calculate the 95% Confidence Interval for all metrics to determine whether the differences between the models are statistically meaningful.

5.3 Experimental Results

The Monte Carlo simulation was conducted over **3,000 iterations** to get statistically meaningful data. The results show that the **Conditional Filtering** model outperforms the Utility-Based approach across all three key metrics.

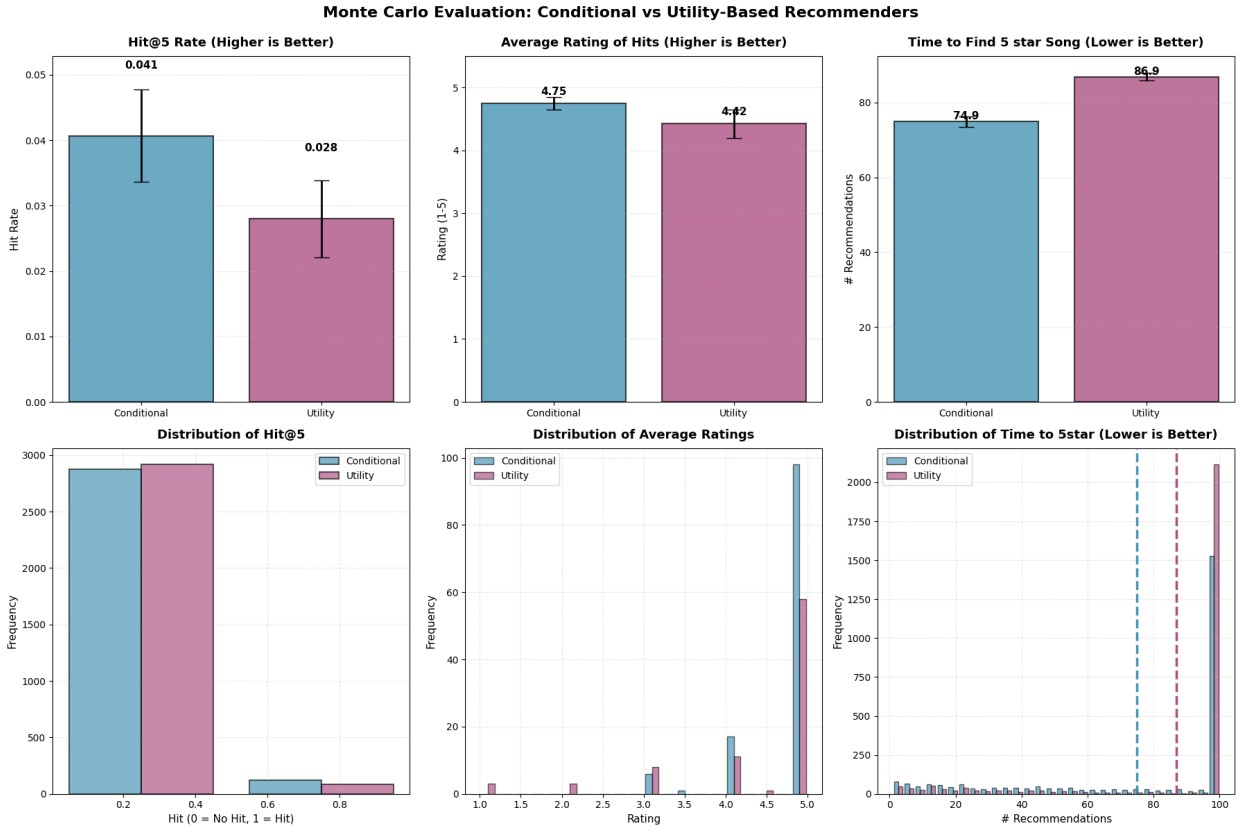

Figure 2: Monte Carlo Evaluation Results (N=3000): Comparison of Conditional vs. Utility models. Error bars represent 95% Confidence Intervals.

Output:

Hit@k:

Conditional Filtering: Sample size: 3000 — Mean: 0.0407 — 95% CI: [0.0336, 0.0477]

Utility-Based Sampling: Sample size: 3000 — Mean: 0.0280 — 95% CI: [0.0221, 0.0339]
Model Comparison: Conditional is better: outperforms by 1.27% (95% CI of diff: [0.40%,2.14%]).
Average Rating:
Conditional Filtering: Sample size: 122 — Mean: 4.7500 — 95% CI: [4.6537, 4.8463]
Utility-Based Sampling: Sample size: 84 — Mean: 4.4226 — 95% CI: [4.1996, 4.6456]
Model Comparison: Conditional is better: outperforms by 0.33 points (95% CI of diff: [0.08,0.57]).
Time-to-5 star:
Conditional Filtering: Sample size: 2624 — Mean: 74.8815 — 95% CI: [73.5594, 76.2035]
Utility-Based Sampling: Sample size: 2624 — Mean: 86.9120 — 95% CI: [85.8098, 88.0142]
Model Comparison: Conditional is better: finds 5-star songs 12.03 recommendations faster (p-value: 0.0000).

5.3.1 Hit Rate (Hit@5)

The **Conditional Filtering** model showed greater performance in the top-5 recommendations.

- **Performance:** Conditional Filtering achieved a mean hit rate of **4.07%** ($Mean = 0.0407$). On the other hand, the Utility model achieved **2.80%** ($Mena = 0.0280$).
- **Significance:** The analysis of mean differences confirms that Conditional Filtering outperforms the Utility model by approximately **1.27%** ($\mu_{conditional} - \mu_{utility} = 1.27\%$). The 95% Confidence Interval of this difference is **[0.40%, 2.14%]**. Since this interval does not include zero, the result is **statistically significant**.

5.3.2 Average Rating of Hits

The conditional model recommends songs that are more suitable to the user's preferences when a song in the test set is found also in the recommendations.

- **Performance:** The conditional model achieved a higher average rating of **4.75** compared to the utility model's **4.42**.
- **Significance:** The 95% confidence interval for the difference in ratings is **[0.08, 0.57]**. This confirms that the conditional model not only finds hits more often but also recommends songs that users rate roughly **0.33 points higher** on average.

5.3.3 Time-to-5$\star$

The conditional model finds a favorite song faster than the utility model.

- **Conditional Model:** Required an average of **74.9** recommendations to find a 5-star song.
- **Utility Model:** Required more searches, around **86.9** recommendations.
- **Significance:** Due to the non-normal distribution of this metric, we compared the result using the Mann-Whitney U test that is used in the part 2. The resulting **p-value is almost zero ($p < 0.001$)**, showing strong statistical evidence that the conditional model is faster.

5.4 Discussion

The experimental results demonstrates that the conditional model performs better in this sparse data environment(in contrary to our belief when we are developing the recommenders). We developed the conditional model with the focus on the user preferences so it's performance will gets better with more data about the user.

The Failure of Global Popularity: The utility model's lower hit rate (2.80%) and higher time to 5 star (86.9) indicates the limitations of global popularity. If a user's taste is slightly distinct, the utility model' recommendations gets irrelevant, often reaching the recommendation limit (100) without finding any test songs.

The Efficiency of Conditional Filtering: By filtering the songs based on user-specific features (e.g., Artist or Genre), it achieves a good of performance on the metrics:

1. **Higher Accuracy:** It finds relevant songs more often (+1.27% Hit Rate).
2. **Higher Quality:** The found songs are liked more (+0.33 Rating).
3. **Higher Efficiency:** It finds user favorites faster (∼12 fewer recommendations).

In conclusion, while data insufficiency remains a challenge (overall hit rates are low), conditional model provides a statistically significant improvement over popularity-based model, offering a more consistent and satisfying user experience.

References

[1] https://en.wikipedia.org/wiki/Monte_Carlo_method

[2] https://www.ibm.com/think/topics/monte-carlo-simulation