The objective of the project itself was to build an accurate classification system that can predict the amount of stars given to a movie by a user. We are not able to utilize deep learning or any type of neural network in this assignment. Using traditional algorithms, we were able to achieve an accuracy rate ranging from 56% to 62%.

With certain tweaks, my code can get up to 62%, however, I have submitted the version that got me 57% and 58%. I am no longer sure if I should even submit this version, because I initially thought that the 62% version was unstable, as it caused many RAM issues for me. However, I am now getting RAM issues from the 58% version, leading me to be extremely confused. I am going to aire on the side of caution and consider the 58% version as my official submission, however I'd just like the reviewers to know that there was room for improvement.

The goal for this assignment was to reach a 68% accuracy. While the goal was not achieved in the particular solutions submitted by me, this project can definitely reach that accuracy, and the code submitted is not too far off. Without further ado, let's dive into some of the specific mechanisms used in this project.

First, since we can't use complicated methods, we need to make sure that we can work with the data. An ample amount of preprocessing is done on the data set to ensure that it plays nice with our code. We have to ensure consistency in the data presented and the data types (int float etc).

Feature selection and scaling were an important part of the project. Adding in more features might increase your accuracy at times, but it could also cause a drastic uptick in processing times and memory consumed. Playing between these boundaries was an important part of the challenge, especially considering the limited consumer-grade hardware that we ran the code on. In my case, it was an M1 Macbook Pro, which usually has enough power for anything I

throw at it. In this case however, due to such a large dataset, there were significant instances of curse words being exclaimed, slamming the lid shut, and a lot of waiting. Ranking the features played an important role in not only the accuracy and noise reduction, but also in determining which features to drop when runtime was of essence.

I played with a few models but ultimately settled on Perceptron simply for its speed. Out of all the models I played with, Perceptron would allow for a high speed and good enough accuracy. The specific parameters of the vectorizer and of the model itself played a huge role in how much RAM was consumed and how long the project ran for, but at the same time, it also played a role in the accuracy. When it became clear to me that I was not going to come close to full points, I decided to shift my approach and hunt for a solution that was stable, reproducible, and hopefully easy to understand.

In terms of tips and tricks, I honestly didn't do much that was too robust or complex. Most of my code follows the exact same patterns of the starter code. The thought process I had was to just try to brute force the problem against GPT and what starter code we were given, alongside what I learned in class. I shifted to an accuracy approach when I finally got the starter code working at a meager 40~ or 50~ reported accuracy, however I quickly realized that it was not sustainable for me to keep a 60~ accuracy if it took forever to run and would usually end in my computer restarting.

I don't really think my approach needs too much explaining but in the interest of fulfilling the requirements, I played carefully with features to ensure a solid model of how to rank the text. I merged the summary and text into one column. I used the TF-IDF Vectorizer to reduce the dimensionality, and I used hstack for a similar purpose. Tweaks were made in max

features variable for vectorizer, and also in max iter and tol in the model until a final solution was reached.

In terms of a general overview of the workflow process and how I reached my solution, it basically started from brute forcing the startercode into a working prototype, albeit slow and inaccurate, and then throwing my head at the keyboard until I eventually reached my final solution. From around 2 AM Sunday to 12 PM on Monday, I did not step foot outside of my room. It was worth it though, as I now have a better understanding of data science.

While I do understand what I worked on and I understand why it works the way it does, I will admit that there was a strong amount of prompt engineering that led me to the solution which I ended up with. I'd like to thank all the posts in the discord, my TA, the class slides, ChatGPT, the professor, and most importantly, whoever is grading this paper. I'm sorry for any mistakes in my essay; English is not my first language. I don't like to use ChatGPT to generate text so I've written this all out and then combed through it with translation tools and other references. Please forgive any errors.